October, 1965 Report ESL-R-248 Copy 46

CIRCAL: ON-LINE ANALYSIS OF ELECTRONIC NETWORKS

by

M. L. Dertouzos and C. W. Therrien

The preparation and publication of this report, including the research on which it is based, was sponsored by the National Aeronautics and Space Administration under Research Grant No. NsG-496 (Part), M.I.T. Project DSR No. 9948. This report is published for information purposes only and does not represent recommendations or conclusions of the sponsoring agency. Reproduction in whole or in part is permitted for any purpose of the United States Government.

Electronic Systems Laboratory
Department of Electrical Engineering
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

ABSTRACT

A method is presented for the on-line simulation of electrical networks. The program generated from such a method can be used as a powerful design tool in man-machine interaction. The method presently treats networks consisting of linear and nonlinear resistors as well as linear energy storage elements and can be extended to nonlinear energy storage elements. The network is described to the computer either by typing elements and nodes to which they connect or by composing the network with a light pen on a cathode ray tube. Voltage and current sources which may be arbitrary functions of time excite the network, and the voltage across any pair of nodes as a function of time is calculated and may be displayed.

The computer model for the network consists of storage-resistor lists representing branches connected to other lists representing nodes.

Solution is performed by decomposing the dynamic problem into a number of static problems for each time interval. Each static problem yields the output and new state of the network on the basis of the excitation and present state. State is determined by the energy-storage elements, and solution of the amnesic problem is accomplished by relaxing node voltages subject to the Kirchoff's current law constraint.

A computer program for use on the Project MAC compatible time-sharing system was written to implement the foregoing method. This preliminary program called CIRCAL-1 produced 200 points of the time solution for moderate sized networks (up to about 8 nodes and 12 branches) in the order of a few seconds.

author

CONTENTS

CONTENTS (Cont.)

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Recent developments in large-scale, time-shared digital computers[*4] have created interest in computer-aided design of electrical networks. Fundamental to the design of such networks is the ability of the digital computer to analyze and the ability of the user to synthesize by evaluating analytical results. This report presents CIRCAL-1, the first version of an electrical-network simulation program called CIRCAL (CIRCuit anALysis), the central aim of which is analysis of unrestricted electrical networks. The terms "analysis" and "simulation" are so commonly used that often they are insufficient for conveying to the reader a clear picture of the objectives of CIRCAL. In order to clarify these objectives, the following description of a typical analysis session between user and computer is presented.

The user, after identifying himself to the Compatible Time-Sharing System (CTSS) types a few key words to load CIRCAL from his files into computer memory. The command "input" is typed and the program is now ready to accept a network description. Holding a light-pen, the user points to a desired position on a cathode-ray-tube (CRT) and presses appropriate push buttons for the type and orientation of the electrical element that he wishes to introduce.[**] This element appears on the CRT in conventional graphic-symbolic form and the user types relevant information concerning the element, such as its value, type, and tolerance. Repeating this process, the user gradually composes on the CRT the entire network under consideration, including sources of excitation. Subsequent typing of the command "analiz" causes the computer to analyze and store the dynamic response at all points of the network. Further commands may then be typed, such as "disply v(i, j)" resulting in a graphical display

---

[*] Superscripts refer to numbered items in the Bibliography.

[**] The same result can be accomplished by typing in descriptions of the network elements.

on the CRT of the voltage between nodes i and j. The user, upon observing v(i,j) may introduce some changes either in topology or in parameters of the given network, by appropriate commands. Successive use of these steps is made until the user is satisfied with the results. Thus far, the session that has been described is typical of CIRCAL-1. A number of future extensions and modifications can be visualized. It is, for example, conceivable that upon satisfactory completion of the foregoing design session, numerically-controlled tools may be instructed to convert automatically the computer design into an actual circuit.

Between CIRCAL-1, however, and the final version of CIRCAL lie a number of progressively more sophisticated versions, satisfying additional objectives. The basic objectives of CIRCAL are as follows:

Communication between designer and computer should be in conventional circuit terminology, and should be as independent as possible of computer programming. A design engineer with no computer experience should be able to learn and use CIRCAL in a matter of minutes; it will be then possible for the designer to concentrate primarily on design issues rather than on operational technicalities.

The designer should be able to edit on-line a given network and to observe results of this editing. By editing here we mean the modification of component values, source values or types, component characteristics, and the insertion (or deletion) of elements. This objective is essential for successful design, especially in cases where formal design procedures are lacking, since it entails a "feedback" mechanism between user and computer.

In addition to conventional elements, such as linear resistors, capacitors, inductors and diodes it should be possible for the user to define and use his own special elements. For example, a nonlinear resistor could be defined by drawing with the light pen the desired nonlinearity on the CRT. Alternatively, that resistor could be defined by specifying the nonlinearity analytically.

There should be a compatibility of the system in all levels of hierarchy. That is, a given network consisting of a number of elements could in turn be called an element, and used as such in the composition and analysis of a still larger network.

Special design aids should be incorporated such as the simulation of environmental changes, aging of components, tolerance variations, etc. For example, by typing a given temperature range it should be possible to obtain and display dynamic responses of the network at the extreme temperatures of that range. The temperature law that elements obey could be inherent to the system or externally adjustable.

Finally the system should have growth capability so that modification and additions can be made on subsystems without affecting adversely the overall system. This growth capability is essential to CIRCAL for the gradual incorporation of the foregoing objectives and for the introduction of any new objectives.

The working foundations of CIRCAL have been established on the basis of the foregoing objectives and the technological constraints imposed by time-sharing systems. It was decided to analyze networks using the state-variable approach. The primary reasons for this selection are the generality afforded by such an approach in analyzing unrestricted networks, and the ease of mapping the electrical network into a computer model. Although usually the state vector representing the network is taken to have the minimum number of dimensions, the approach taken in CIRCAL-1 calls for one state vector dimension per energy storage element. Reasons for this decision are the great ease of constructing a computer model from the network and the physical relation of the state-vector components to the network elements. Through the state-variable approach, the actual dynamic behavior of the network is decomposed in a time sequence of static problems, where the solution of each static problem is used for revising the state used at the next static problem and for determining the output of the network. Subsequent solution of the static problem, equivalent to solution of a set of simultaneous nonlinear algebraic equations, may be accomplished in a number of ways. In CIRCAL-1 an iterative-relaxation approach was chosen, where fictitious node voltages are relaxed to their actual values on the basis of the current sum at each node. This approach converges to a solution in a finite number of iterations, dependent on network topology and component values, for a class of amnesic nonlinearities. The reasons for selecting this approach in CIRCAL-1 are its independence from network topology and its ease of implementation.

Later versions of CIRCAL will use other approaches for solution of the static problem, and the results will be evaluated experimentally. Experimental evaluation of these approaches is necessary since relative estimates of computer time can be theoretically bounded, but these bounds are typically quite loose, as shown later in the report.

In addition to these fundamental decisions for CIRCAL-1 various other secondary decisions were taken on types of sources, elements, and so forth, in order to expedite evolution of a working program. These decisions are explained in more detail as they are presented in the report.

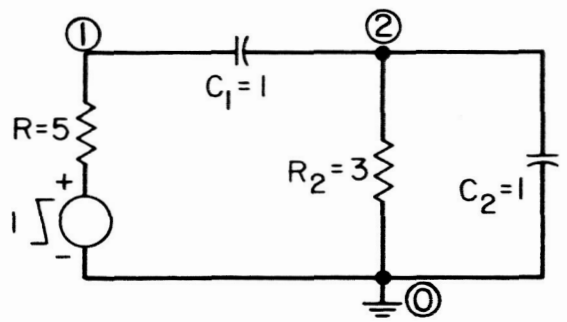
To summarize, CIRCAL-1 analyzes linear, time-invariant R L C networks with independent voltage or current sources that are steps or sinusoids. Network size is limited to 50 branches and 20 nodes, and networks can be given to the computer either graphically or through a typer. Outputs are available as either voltage-versus-time lists on a typer, or graphs on either the CRT or typer. Observe that the foregoing limitations are not applicable to all subsystems of CIRCAL-1 but only to the present overall working version. For example, the procedure solving the static problem can be applied to nonlinearities of a certain class and the state-variable approach is applicable to general nonlinear energy-storage elements. A graphical input-output program is used to communicate with CIRCAL-1 through the Electronic Systems Laboratory Display Console at Project MAC. A detailed report on this activity is forthcoming under the title "Graphical Communication for Electrical Network Simulation."

Throughout CIRCAL, the internal computer model or data structure is established in almost one-to-one correspondence with the actual network. Thus, resistors, inductors, capacitors and other elements are represented within the computer with computational blocks which carry all the relevant information about their corresponding real elements. These blocks are in turn interconnected through a convenient addressing system in correspondence with the real element interconnection of the given network. Thus, additions, deletions or modifications of elements in the network correspond to identical operations on the computational blocks. Moreover, the two basic algorithms (dynamic-to-static problem decomposition and static problem solution)

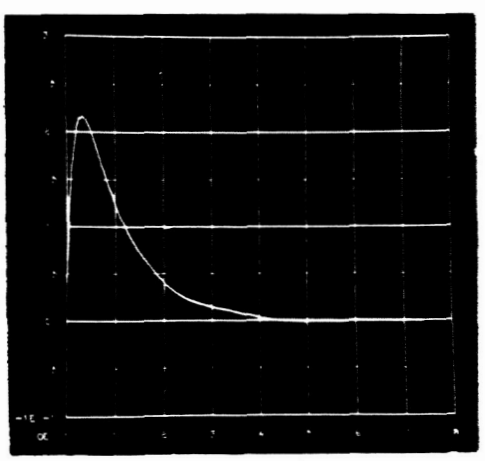
are applicable to any topological structure of the foregoing blocks, so that the ultimate fundamental limitations of CIRCAL shall be due to computer memory and time constraints, rather than to programming.

Example of an Application:

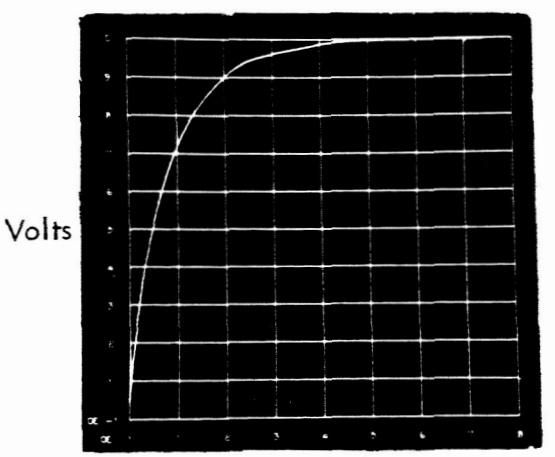
As an example of a network analyzed by CIRCAL-1 consider the circuit of Figure 1.1(a). The network is described to the computer, branch by branch. Then the program is instructed to compute the dynamic response at all the nodes, and to plot the voltage across nodes 1 and 2 and across nodes 2 and 0. The results as displayed on the CRT, are shown in Figure 1.1(b) and 1.1(c).



(a) The network



(b) The voltage across C_1



(c) The voltage across C_2

Fig. 1.1 Step Response of an RC Network Computed by CIRCAL-1

CHAPTER II

METHOD OF ANALYSIS

A. INTRODUCTION

This chapter presents a method for the dynamic analysis of networks consisting of capacitors, inductors, nonlinear resistors, and independent sources. The dynamic problem is represented by a series of static approximations over time intervals small compared with the time over which the network solution changes appreciably. Solution of the static problem at a given time interval entails knowledge of the state of the network at the previous time interval. A new state is then computed, to be used in the static solution of the next time interval. Solution of the static problem is accomplished through a relaxation* method, where an arbitrary initial set of node voltages is successively perturbed until equilibrium within a tolerable error is achieved. Updating the state of the network for the next time interval is accomplished through straightforward numerical integration. State is represented here as the set of capacitor voltages, inductor currents, and parameters characterizing other energy storage elements. Observe that this representation does not necessarily result in the smallest number of independent state variables. This slight sacrifice of minimality is well justified by the resulting ease of modeling the given network within the computer.

Section B discusses linear amnesic** networks, and describes a relaxation method for their analysis. The results of that section are extended in Section C to certain types of nonlinear amnesic networks. Finally, Section D treats the dynamic non-amnesic network problem and shows how it can be formulated in terms of a series of amnesic network problems.

---

* Here the term relaxation refers to the method of Section B. In the literature, this term is generally used to describe a method proposed by Southwell[26] for the analysis of mechanical structures.

** We will borrow a term from Zimmerman and Mason[27] to refer to elements without memory. The term non-amnesic will be used to mean elements with memory.

The method of Section B was independently derived by the authors following a similar perturbation technique used in threshold logic.[7] A similar method, however, has been explored by Katzenelson and Seitelman[17] with the electrical network problem in mind and probably by others in related applications. A similar method for solving amnesic networks (not necessarily electrical) has been presented by Birchoff and Diaz[2] employing the relaxation technique of Southwell.[26] In addition, there exist more general schemes for solving systems of linear equations that employ related perturbation relaxation and iteration techniques (see e.g. Fadeeva[11]).

B. THE LINEAR AMNESIC PROBLEM

Consider a linear passive amnesic network with N+1 nodes. Let the network be excited by time-invariant ideal voltage sources inserted in the branches and by ideal current sources connected across pairs of nodes.* One of the nodes will be defined as datum (zero volts), and an arbitrary set of node potentials $\{e_j^o\}$** $j = 1, 2, \ldots, N$ will be assigned to the remaining N nodes. If $\{E_j\}$ $j = 1, 2, \ldots, N$ is the set of node potentials required for equilibrium of the network then it is desired to successively perturb $\{e_j^o\}$ through $\{e_j^1\}, \{e_j^2\}, \ldots,$ where $\{e_j^m\}$ denotes the set of node potentials at the mth iteration, until a set of node potentials $\{e_j^p\}$ is found sufficiently close to $\{E_j\}$ to satisfy the error bound

$$\sum_{j=1}^{N} (E_j - e_j^p)^2 < \epsilon \qquad (2.1)$$

where ϵ is a tolerable upper bound on the sum-square error.

For convergence of the iterative process we require a reduction of the sum-square error between successive iterations, i.e.,

$$\sum_{j=1}^{N} (E_j - e_j^{m+1})^2 < \sum_{j=1}^{N} (E_j - e_j^m)^2 \qquad (2.2)$$

for each m, such that $m < p$.

---

* Guillemin[14] calls these pliers and soldering-iron entries respectively.

** An ordered set $\{x_j\}$, $j = 1, 2, \ldots, N$ will be also denoted by the N-dimensional column vector $\underline{x}$.

Let ΔI_i^m be the algebraic sum of the currents flowing into the ith node, at the mth iteration, when the set of applied node potentials is $\{e_j^m\}$, $j = 1,2,\ldots,N$. Define $\{e_j^{m+1}\}$ as follows:

$$e_j^{m+1} \triangleq e_j^m + K\Delta I_j^m, \quad j = 1,2,\ldots,N \tag{2.3}$$

where K is a scalar, hereafter called the convergence constant. Based on the foregoing definitions, the following theorem is established.

Theorem 2.1

Inequality 2.2 is satisfied for values of K in Eq. 2.3 within a least upper bound K_v, and a greatest lower bound 0. The proof of this theorem is presented in Appendix A. In that appendix it is seen that the least upper bound K_v is given by

$$K_v = \frac{2\underline{\Delta I}^T [G^{-1}] \underline{\Delta I}}{\underline{\Delta I}^T \underline{\Delta I}} \tag{2.4}$$

where $\underline{\Delta I}$ is a vector representation of the ordered set $\{\Delta I_j\}$, $j = 1,2,\ldots,N$, $[G]$ is the conductance matrix of the network, and the superscripts T and -1 indicate transpose and inverse respectively. Evaluation of this bound involves more operations than direct solution of the network,

$$\underline{E} = [G^{-1}] \underline{Is} \tag{2.5}$$

where $\underline{Is}$ is a vector representing the independent sources of network (see also Appendix A). Moreover, the progress of such an iteration procedure cannot be conveniently monitored, since Inequality 2.2 includes the equilibrium potentials E_j, which are naturally unknown during the iteration process. It is possible to compute the quantities $\sum_{j=1}^{N} (E_j - e_j^m)^2$ of Inequality (2.2) through use of the inverse conductance matrix as follows:

$$\underline{E} - \underline{e}^m = [G^{-1}] \underline{\Delta I}^m \tag{2.6}$$

This approach, however, suffers from the same disadvantage, that is the computation of $[G^{-1}]$. On the other hand, it is possible to establish a criterion similar to Inequality 2.1 on the norm of $\underline{\Delta I}$ and show that satisfaction of this criterion implies convergence. This will be shown in the following:

Let it be required that after q iterations the set $\{\Delta I_j^q\}$, $j = 1,2,\ldots,N$ satisfies the bound

$$\sum_{j=1}^{N} (\Delta I_j^q)^2 < \delta \qquad (2.7)$$

The left hand side of Inequality 2.7 will be called the current error and δ, a positive scalar, will be termed the current error tolerance. It is shown in Appendix B that if a value $\delta \leq \frac{\epsilon}{\|G^{-1}\|}$ is chosen where $\|G^{-1}\|$ is the norm* of $[G^{-1}]$, then Inequality 2.7 implies Inequality 2.1. As before, arbitrary node potentials $\{e_j^o\}$ will be successively perturbed such that

$$\sum_{j=1}^{N} (\Delta I_j^{m+1})^2 < \sum_{j=1}^{N} (\Delta I_j^m)^2 \qquad (2.8)$$

for $m<q$ until Inequality 2.7 is satisfied. Let $\{e_j^{m+1}\}$ be related to $\{e_j^m\}$ by Eq. 2.3. Then the following theorem is established.

Theorem 2.2

Inequality 2.8 is satisfied for values of K in Eq. 2.3 within a least upper bound K_I and a greatest lower bound 0.

Proof

Using matrix notation, Inequality 2.8 can be rewritten as

$$[\underline{\Delta I}^{m+1}]^T \underline{\Delta I}^{m+1} < [\underline{\Delta I}^m]^T \underline{\Delta I}^m \qquad (2.9)$$

Define the function U(K) by

$$U(K) \triangleq [\underline{\Delta I}^m]^T \underline{\Delta I}^m - [\underline{\Delta I}'^{m+1}]^T \underline{\Delta I}^{m+1} \qquad (2.10)$$

Inequality 2.9 requires U(K) to be positive for $m<q$. Substituting Eq. 2.6 into Eq. 2.10, using Eq. 2.3 and rearranging yields:

$$U(K) = -K^2 \underline{\Delta I}^T [G]^2 \underline{\Delta I} + 2K \underline{\Delta I}^T [G] \underline{\Delta I} > 0 \qquad (2.11)$$

---

* The norm $\|A\|$ of a matrix A is defined as $\|A\| \triangleq \max_{\|\underline{X}\|=1} \|AX\|$

where $\|\underline{X}\|$ is the norm of X, defined as $\|X\| \triangleq \underline{X}^T \underline{X}$.

where the superscript m has been dropped for convenience and where we have used the fact that the conductance matrix $[G]$ is symmetric, i.e.,

$$[G]^T = [G] \text{ and } [G]^2 \triangleq [G]\,[G]$$

Since $[G]$ and therefore also $[G]^2$ is positive definite, the quadratic forms $\underline{\Delta I}^T[G]^2\,\underline{\Delta I}$ and $\underline{\Delta I}^T[G]\,\underline{\Delta I}$ are positive quantities for all $\Delta I \neq 0$. A plot of U(K) versus K is shown in Fig. 2.1. From either Eq. 2.11 or Fig. 2.1 it can be seen that the greatest lower bound is 0 and the least upper bound K_I is

$$K_I \triangleq \frac{2\,\underline{\Delta I}^T[G]\,\underline{\Delta I}}{\underline{\Delta I}^T[G]^2\,\underline{\Delta I}} \tag{2.12}$$

Q.E.D.

Evidently, fastest convergence occurs for $K_{opt} = \frac{1}{2} K_I$, i.e.,

$$K_{opt} = \frac{\underline{\Delta I}^T[G]\underline{\Delta I}}{\underline{\Delta I}^T[G]^2\underline{\Delta I}} \tag{2.13}$$

From Eq. 2.13 it is seen that evaluation of K_{opt} involves certain matrix operations which depend on the network topology. These operations are not as time consuming as matrix inversion since they involve use of matrices $[G]$ and $[G]^2$. Nevertheless, since K_{opt} changes in general with each iteration (each different $\underline{\Delta I}^m$), the investment of computational effort to evaluate K_{opt} may be wasteful. In CIRCAL-1 a search based on Newton's method is used to approximate initially K_{opt}. This method is executed once at the beginning of the iterative process and the resultant value of K_{opt} is retained for succeeding iterations, unless convergence cannot be achieved. In this case a new value of K_{opt} is computed. More details on this method are presented in Chapter III.

Let us examine now certain issues pertaining to the convergence of the iteration procedure that has been described. The questions of greatest interest here are

a) Does the process converge in a finite number of iterations ?

b) What is the fastest rate of convergence?

c) How does this approach compare with classical matrix-inversion?

In order to answer these questions consider the function V(m), defined below, which establishes a measure of convergence between adjacent iterations in an N-dimensional vector space.

$$V(m) \triangleq \frac{|\Delta I^m|^2 - |\Delta I^{m+1}|^2}{|\Delta I^m|^2} \qquad (2.14)$$

V(m) is the fractional change of the current error vector between the mth and m+1th iterations when the optimum value of K, K_{opt} is used in Eq. 2.3.

Substitution of Eq. 2.3 and 2.6 into 2.14 yields after some rearranging

$$V(m) = \frac{2K\underline{\Delta I}^T[G]\,\underline{\Delta I} - K^2\underline{\Delta I}^T[G]^2\,\underline{\Delta I}}{|\underline{\Delta I}|^2} \qquad (2.15)$$

where the superscript m has been dropped from $\underline{\Delta I}$ for convenience.

Substituting in Eq. 2.15 K_{opt} from Eq. 2.13 yields after some rearranging

$$V(m) = \frac{1}{|\underline{\Delta I}|^2}\,\frac{(\underline{\Delta I}^T[G]\,\underline{\Delta I})^2}{(\underline{\Delta I}^T[G]^2\underline{\Delta I})} = \frac{(\underline{\Delta I}^T[G]\,\underline{\Delta I})^2}{|\underline{\Delta I}|^2\,|[G]\underline{\Delta I}|^2} \qquad (2.16)$$

Observe that if numerator and denominator of Eq. 2.16 are multiplied by K_{opt}^2, then V(m) can be interpreted as the cosine squared of the "angle", θ, formed between the vectors $\underline{\Delta I}^m$ and $K_{opt}[G]\,\underline{\Delta I}^m$. The latter, however, is the vector perturbing $\underline{\Delta I}^m$ into $\underline{\Delta I}^{m+1}$. These observations are illustrated in Fig. 2.2. Angle θ may be further interpreted as formed by the direction in which $\underline{\Delta I}^m$ is perturbed and the direction in which $\underline{\Delta I}^m$ should be perturbed for most rapid termination of the procedure.

We proceed to show that

$$V(m) = \cos^2\theta \qquad (2.17)$$

is lower-bounded by a positive constant dependent on the network under consideration. The numerator of Eq. 2.16 is a quadratic form of a positive-definite symmetric matrix with a minimum at

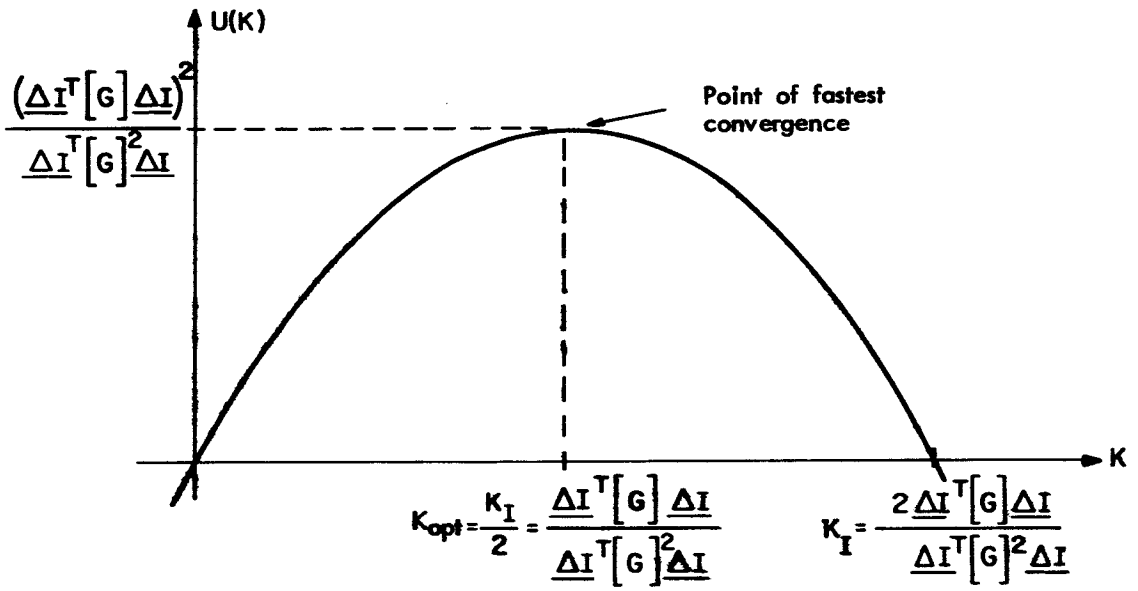


Fig. 2.1 Convergence Curve for Linear Amnesic Network

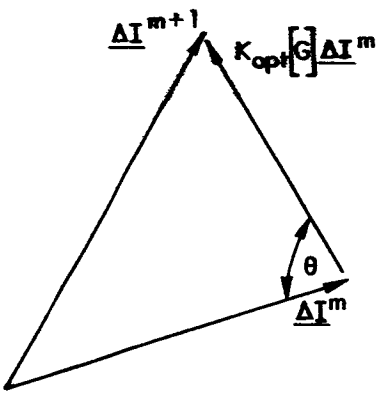


Fig. 2.2 Geometrical Interpretation of V(m)

$\underline{\Delta I} = \underline{0}$, (equilibrium). For any fixed deviation ($\underline{\Delta I}$) from equilibrium, the quadratic form is bounded as follows:

$$\underline{\Delta I}^T [G] \underline{\Delta I} \geq \lambda_{min} \quad |\underline{\Delta I}|^2 \tag{2.18}$$

where λ_{min} is the smallest* eigenvalue of $[G]$. The denominator is upper bounded by

$$|\Delta I|^2 \; |[G]\Delta I|^2 \leq |\Delta I|^4 \lambda_{max}^2 \tag{2.19}$$

where λ_{max} is the largest eigenvalue and hence the norm of $[G]$.

Eq. 2.18 and 2.19 establish the following lower bound on V(m)

$$V(m) = \cos^2\theta \geq \frac{\lambda_{min}^2}{\lambda_{max}^2} \tag{2.20}$$

It is desirable to establish a relationship between the extremal eigenvalues λ_{min}, λ_{max} and the network structure. Such a relationship exists in the form of the following very loose bound[19]

$$\lambda_{min} > \frac{1}{NR_{tot}} \tag{2.21}$$

$$\lambda_{max} < NG_{tot}$$

where R_{tot} and G_{tot} are the sums of all the resistances and all the conductances in the network of N+1 nodes. Based on Eq. 2.21 the bound of Eq. 2.20 becomes

$$V(m) = \cos^2\theta \geq \frac{1}{N^2 R_{tot} G_{tot}} \tag{2.22}$$

We shall compare this bound with experimental data in Chapter IV, Section E. For the time being it is sufficient to note that this is not a tight bound.

From Eq. 2.14 and Eq. 2.17 it follows that

$$|\Delta I^{m+1}| = |\Delta I^m| \sqrt{1 - \cos^2\theta} \tag{2.23}$$

Letting the initial root current error $|\Delta I^o|$ be bounded by αI_f where α is a fraction and I_f is the magnitude of full scale current. More-

*Recall that all eigenvalues of $[G]$ are positive.

over, let βI_f be the maximum tolerable root current error, i.e., the value of $|\Delta I^p|$ at which the iteration process terminates. Recursive use of Eq. 2.23 and the foregoing definitions of α, β and I_f yields for the number of iterations, q:

$$q \leq \frac{2 \log \beta/\alpha}{\log(1 - \cos^2\theta)} \tag{2.24}$$

Since $\cos^2\theta$ is lower bounded by Eq. 2.20, $1 - \cos^2\theta$ is upper bounded and therefore q in Eq. 2.24 is lower bounded. This answers the first question that was raised earlier, that is it confirms convergence of the procedure in a finite number of iterations.

For small values of $\cos^2\theta$, and β/α the number of iterations can be approximated by

$$\max q \cong \frac{2}{\cos^2\theta} \tag{2.25}$$

Substituting Eq. 2.20 into Eq. 2.15 yields

$$\max q < 2 \frac{\lambda^2_{max}}{\lambda^2_{min}} \tag{2.26}$$

Evidently, the spread of eigenvalues of [G] determines convergence rate. An accurate estimate of convergence rate as a function of network size is unfortunately not possible from Eq. 2.21. For example, as shown in Chapter IV, Section E a set of networks that have been analyzed by CIRCAL-1 yielded a growth of the number of iterations proportional to $N^{3/2}$, whereas the bound of Eq. 2.26 predicted a growth of N^8 for these networks.

To summarize, the answer to the second question that was raised in the foregoing is explicit in terms of the extremal eigenvalues of the network but not in terms of network size.

Solving the network by classical matrix inversion requires a computational time which can be divided into two parts. The first part consists of converting the topology of the network into a matrix [G] and the second part involves inversion of [G] to obtain $[G^{-1}]$ and use of $[G^{-1}]$ to obtain the solution. The former of these tasks requires a time proportional to the number of nodes, N, while the

latter requires a time proportional to N^3. Matrix inversion may be conducted only once in the case of a linear time-invariant network between network modifications.* However, in the case of nonlinear networks, matrix inversion may be required for every new value of the state vector.

C. APPLICATION OF THE RELAXATION METHOD TO NONLINEAR AMNESIC NETWORKS

In this section, it will be shown that the method described in section B can also be applied to a class of nonlinear amnesic networks.

Consider a connected network with N+1 nodes containing only amnesic elements and independent voltage and current sources. Let the i-v (current-voltage) characteristics of each nonlinear element satisfy the following constraint:

$$0 < M_L \leq \frac{i_p - i_{p'}}{v_p - v_{p'}} \leq M_U < \infty \tag{2.27}$$

where p and p' are any two points on the i-v characteristic. It can be shown that this condition (related to the familiar Lipschitz conditions) guarantees the existence of a unique solution for the network.[8]

Following the notation of the previous section, let $\{e^m\}$ represent the set of node potentials at the mth iteration. Proceeding as before,

$$\underline{e}^{m+1} = \underline{e}^m + K\underline{\Delta I}^m \tag{2.3}$$

where $\underline{\Delta I}^m$ is the vector, elements of which are the sum of the currents at each node, i.e.,

$$\underline{\Delta I}^m = G[\underline{E} - \underline{e}^m]$$

Consider now a typical nonlinear element satisfying the foregoing condition and illustrated in Fig. 2.3. Let point p on this curve be defined by the voltage, V^m, across the element, i.e.,

$$V^m = e_i^m - e_j^m$$

where the element is connected between the ith and the jth nodes of the network. Applying the algorithm of Eq. 2.3 for the network,

---

* Adding or deleting elements or changing their value.

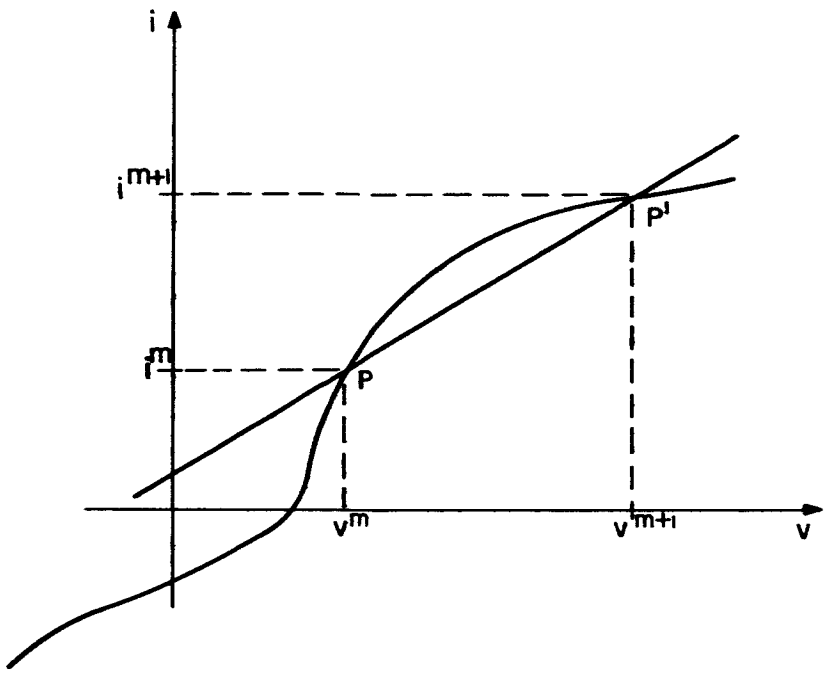


Fig. 2.3 i-v Characteristic of a Typical Nonlinear Element

defines a new point p' determined by

$$v^{m+1} = e_i^{m+1} - e_j^{m+1}$$

If a straight line is passed through points p and p', and the nonlinearity of Fig. 2.3 is substituted with a linear element (resistor and source) having that straight-line as its i-v relationship, then starting from point p, the rule of Eq. 2.3 gives also rise to p' for this new network. In other words, so far as the transition from the mth to the m+1th iteration is concerned, the algorithm 2.3 cannot distinguish whether the nonlinearity or its linear substitute is present in the network. Consequently, the behavior of the current error between adjacent iterations can be obtained from the actual network with appropriate fictitious linear substitutes in place of all the nonlinear elements as illustrated in Fig. 2.3 for one such element. Hereafter, we shall call the actual network nonlinear and the fictitious one linear. The nonlinearities of the network, however, are subject to a condition which guarantees that any two points p and p' on their i-v characteristics will be connected with

a straight line of bounded slope. Hence the conductance of the linear network will have positive, real, bounded eigenvalues between any two iterations. Consequently, the iterative process will converge in a finite number of iterations, determined by the minimum and maximum eigenvalues respectively from the set of all eigenvalues of the fictitious linear-network conductance matrices formed at every iteration.

Observe that the foregoing condition is sufficient, but not necessary, for convergence of the process. It is for example possible to have a small region of negative resistance, in a certain i-v characteristic, if that region is never entered by $\{e^m\}$ in the course of iteration.

D. FORMULATION OF THE GENERAL NON-AMNESIC PROBLEM

It is a known result of the State-space approach[27] to the analysis of nonlinear systems that the entire past history of the inputs $\underline{x}(t)$ up to time t and the initial conditions of a system can be adequately represented for purposes of future analysis in terms of the state vector $\underline{s}(t)$ or state of the network. Generally, the output vector, $\underline{y}(t)$, and the way, $\dot{\underline{s}}(t)$, in which $\underline{s}(t)$ changes are given as amnesic functions of the state and of the inputs, that is

$$\underline{y}(t) = f[\underline{s}(t)]$$

$$\dot{\underline{s}}(t) = g[\underline{s}(t), \underline{x}(t)] \qquad (2.28)$$

Typically, the dimensionality of $\underline{s}(t)$ is the number of independent differential equations describing the network. This number can be obtained from a knowledge of the network topology and involves, in general, some processing. In the following, we establish as state vector $\underline{s}(t)$ the collection of capacitor voltages and inductor currents if capacitors and inductors are the only energy-storage elements. Thus, the state vector $\underline{s}(t)$ is redundant in the sense that it may have more elements than a minimal state vector representation. On the other hand, the one-to-one correspondence established between each network energy storage element and a component of the state vector, facilitates the formation of a direct computer model for the network and makes possible easy modification of that model corresponding to network modifications.

This approach can be regarded as representing each non-amnesic element by an amnesic element with an updatable "state" $s_i(t)$. The network at each instant of time is then completely characterized by its state vector $\underline{s}(t)$, which has a dimension for each non-amnesic element. Since this new network is amnesic, the results of Sections B and C can be applied to obtain a solution at each instant of time.* The states of the nonamnesic elements are then updated, and the solution is computed for the next time instant.

As an example of this approach, consider the i(t) - v(t) relationship defining an inductor

$$i(t) = \frac{1}{L} \int_{-\infty}^{t} v(\tau)\, d\tau \tag{2.29}$$

Let t_n and t_{n-1} be two values of the parameter τ Δt seconds apart, i.e.

$$t_n = t_{n-1} + \Delta t \tag{2.30}$$

Then from Eq. 2.29 we have

$$i(t_n) = \frac{1}{L} \int_{-\infty}^{t_{n-1}} v(\tau)\, d\tau + \frac{1}{L} \int_{t_{n-1}}^{t_n} v(\tau)\, d\tau$$

$$= i(t_{n-1}) + \frac{1}{L} \int_{t_{n-1}}^{t_n} v(\tau)\, d\tau \tag{2.31}$$

The last term in Eq. 2.30 is the area under the curve $v(\tau)$ vs. τ from $\tau = t_{n-1}$ to $\tau = t_n$ as can be seen in Fig. 2.4. This area can be approximated by the area of a trapezoid with bases $v(t_{n-1})$ and $v(t_n)$ and altitude Δt shown in Fig. 2.4. Using this approximation, Eq. 2.30 becomes

---

* Provided that the nonlinearities are continuous and strictly monotonic in the i-v plane.

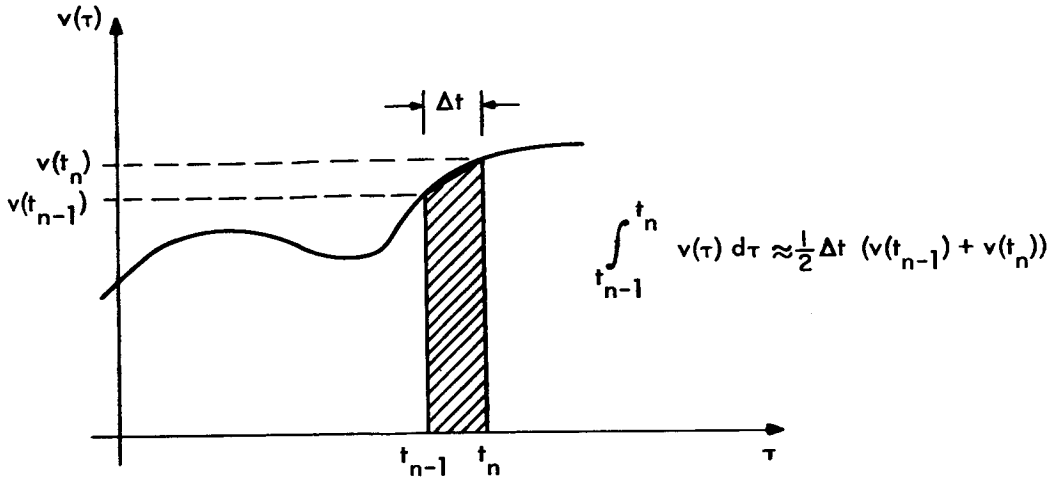
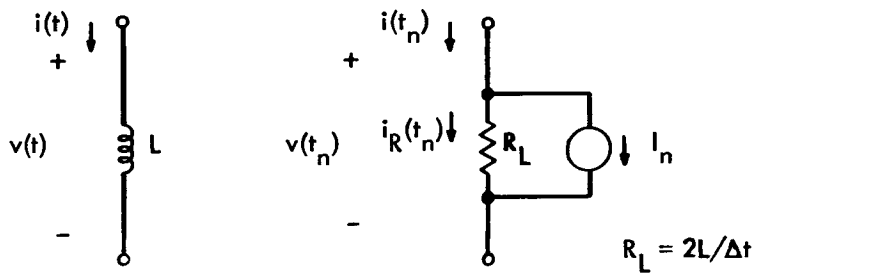
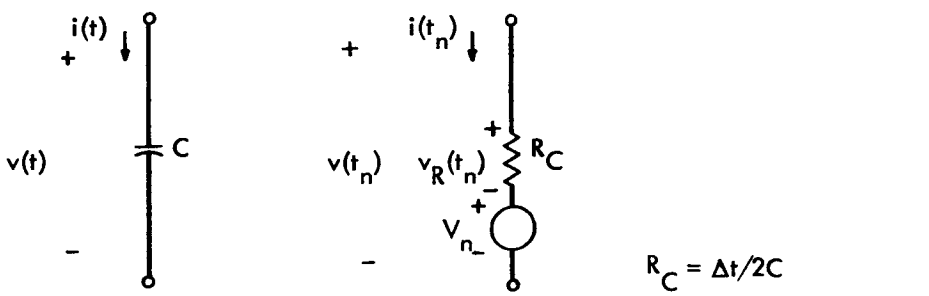


Fig. 2.4 Trapezoidal Approximation to Area under the Curve



(a) Inductor Model



(b) Capacitor Model

Fig. 2.5 Amnesic Models for Inductor and Capacitor

$$i(t_n) \cong i(t_{n-1}) + \frac{1}{L} \cdot \frac{1}{2}\left(v(t_{n-1}) + v(t_n)\right)\Delta t \tag{2.31}$$

If $i_R(t_n)$ is defined as

$$i_R(t_n) \triangleq \frac{\Delta t}{2L} v(t_n) \tag{2.32}$$

then Eq. 2.31 can be written as

$$i(t_n) = i(t_{n-1}) + i_R(t_{n-1}) + \frac{\Delta t}{2L} v(t_n) \tag{2.33}$$

It is assumed that the quantity $i(t_{n-1})$ and the quantity $i_R(t_{n-1})$ defined by Eq. 2.32 are known. Now define

$$I_n \triangleq i(t_{n-1}) + i_R(t_{n-1}) \tag{2.34}$$

and

$$R_L \triangleq \frac{2L}{\Delta t} \tag{2.35}$$

Using these definitions, Eq. 2.33 becomes

$$i(t_n) = I_n + \frac{1}{R_L} v(t_n) \tag{2.36}$$

This equation describes the terminal characteristics of a branch consisting of a linear resistor R_L in parallel with a current source I_n (see Fig. 2.5(a)). If Eq. 2.35 is substituted into Eq. 2.32 and the result is compared with Fig. 2.5(a), the quantity $i_R(t_n)$ defined in Eq. 2.32 can be identified as the current flowing through resistor R_L. This interpretation will be useful in future developments.

A model for the capacitor can be similarly derived, and is shown in Fig. 2.5(b). The terminal relations are given by

$$v(t_n) = V_n + R_C\, i(t_n) \tag{2.37}$$

where

$$R_C \triangleq \frac{\Delta t}{2C} \tag{2.38}$$

$$V_n \triangleq v(t_{n-1}) + v_R(t_{n-1}) \tag{2.39}$$

and where $v_R(t_n)$ is defined by

$$v_R(t_n) \triangleq \frac{\Delta t}{2C} i(t_n) = R_C \; i(t_n) \tag{2.40}$$

and can be interpreted as the voltage across the resistor in the amnesic model.

In most cases where a lumped parameter representation of a physical phenomenon is employed, the memory is represented entirely by capacitors and inductors. However, other non-amnesic elements can be handled in a manner similar to our treatment of the inductor and capacitor. Nonlinear, time-varying, two-terminal energy storage elements can be easily treated, provided that their state representation, usually in differential equation form, is first converted to a difference form

$$v(t_n) = f\left[v(t_{n-1}), i(t_{n-1}) \right] + R\left[v, i, t_n \right] i(t_n)$$

(or the dual of this) where v and i are the terminal voltage and branch current and f and R are arbitrary single-valued functions of their arguments.

CHAPTER III

COMPUTER PROGRAM

A. INTRODUCTION

In this chapter, CIRCAL-1, a computer program for analyzing networks based on the methods of Chapter II, is described. The program is capable of analyzing any connected RLC network (having up to 20 nodes and 50 branches)* with independent voltage and current source excitations. In the present version of the program, these sources must be either sinusoids or steps.** Although the program presently treats only linear networks, modifications can easily be made to include diodes and other nonlinear elements satisfying the conditions set forth in Chapter II. The circuit analysis package was intended for use on the Project MAC time-shared IBM 7094 computer and associated graphical displays, where the user can control the various phases of execution.[4] The programs were written in the AED-0 language,[24] an extended version of ALGOL-60,[20] developed at the Electronic Systems Laboratory, M.I.T.

B. BASIC OPERATION

The computer implementation consists of a main supervisory program CIRCAL† and an associated group of subroutines (see Fig. 3.1). The user when seated at one of the Project MAC consoles gives commands to CIRCAL that enable him to describe and process a circuit. The commands and corresponding computer action are listed in Table 3.1.

---

* The program could be dimensioned to handle much larger networks, however this dimensioning seemed adequate for a preliminary study.

** Since time was short these were thought to be the most useful excitations. Other waveforms such as square, triangular waves, or arbitrary signals introduced as graphical input by the operator can easily be appended.

† The name CIRCAL is used interchangeably both for the family of network-analysis programs, and for the supervisory program within CIRCAL-1.

Table 3.1

List of Commands

Command	Machine Action
input	accepts topological description of network
analiz	computes dynamic solution for entire network
print v(n1, n2)	prints numerical values of the voltage from node n1 to n2 as a function of time
plot v(n1, n2)	plots a graph of the voltage from n1 to n2 on the typewriter
disply v(n1, n2)	plots a graph of the voltage from n1 to n2 on the crt display
erase	removes graph from crt display
quit	ends execution (removes CIRCAL from core)

The "Input" command causes a transfer to the subroutine TOPO. This subprogram accepts descriptions of the network branches from the subroutine INPT and builds the network model, or data structure, in computer memory. When the model has been built, control is returned to CIRCAL. If the user now gives the command "Analiz", the following sequence of events takes place: Control is transferred to DYSOL which sets up the dynamic problem as an amnesic problem over the first interval of time. Control is then passed on to MNIP, which applies the iterative method of Chapter II to compute the amnesic network solution and returns control to DYSOL. Here, the solution for the first time interval is stored, and the state of the network is updated for processing at the next time interval. Control is again passed on to the MNIP routine. This reciprocating motion between DYSOL and MNIP is continued until the solution at all time intervals is computed. Control then returns to CIRCAL. A command of "Print $v(n_1, n_2)$", "Plot $v(n_1, n_2)$" or "Disply $v(n_1, n_2)$" causes the voltage across nodes n_1 and n_2 to be respectively listed, plotted on the typewriter, or displayed on the cathode ray tube.

Two other commands not directly associated with the processing of the network are "Erase" and "Quit". The command "Erase" causes a picture currently being displayed on the CRT to be removed. Typing "Quit" removes CIRCAL and its associated subprograms from memory.

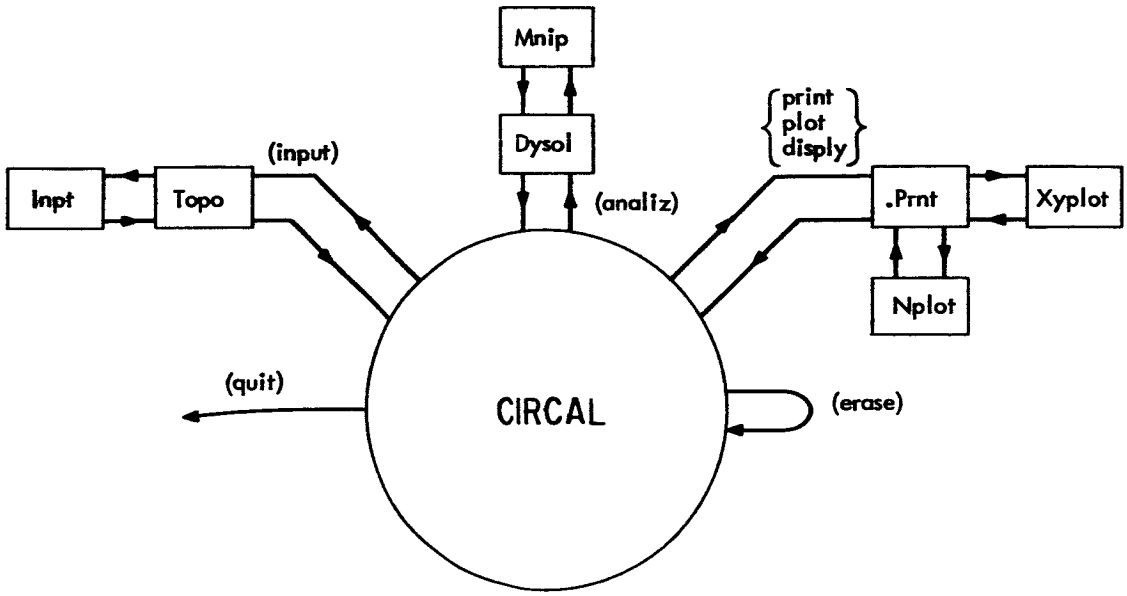


Fig. 3.1 Organization of the Circuit Analysis Program

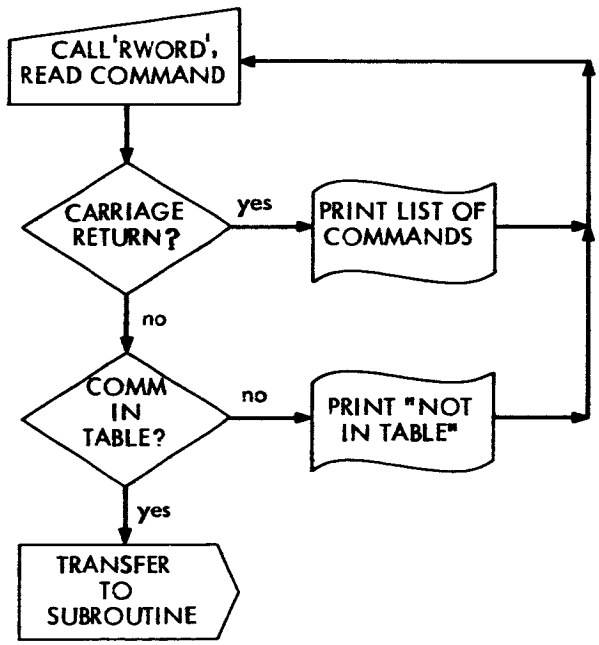


Fig. 3.2 Simplified Flow Chart of the Main Supervisory Program CIRCAL

A simplified flow diagram for the CIRCAL program is presented in Fig. 3.2. The commands are read through a subroutine called RWORD which loads BCD "items" (groups of characters separated by a space, parenthesis or comma) typed on-line into contiguous storage buffers. If a carriage return is the only item read, then CIRCAL will type a list of the available commands and will then ask for the next command. Otherwise the first item read is taken to be the command, and the command table is searched. If the item is not found in the command table, then the program will type the item back to the user, telling him it is not a command, and asking for the next command. If the item read is found in the command table, a transfer will be made to the subprogram that provides the desired action.

C. THE MODELING PLEX

1. General Considerations

Recall from Chapter I that our approach to circuit analysis is to build a model or "data structure" for the network consisting of computational blocks of storage with input and output, representing the branch elements, connected to other blocks of storage assuming the role of nodes. The actual computer implementation departs from this description in just one respect. Instead of providing each simulated element with its own input-output mechanism, each block is given a characteristic "identifier" and an external operator is provided which examines the identifier of each block and supplies the appropriate input-output relation. This approach saves core space since it eliminates unnecessary repetition of computer instructions. The combination of data structure and operator forms what is called the modeling plex for the network.

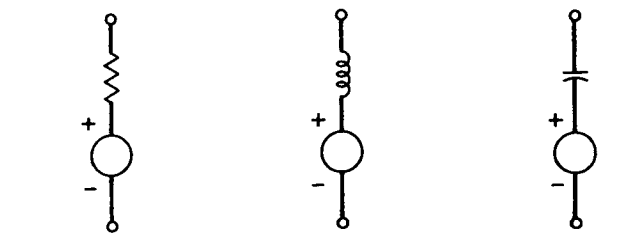
The modeling plex is simplified in two respects if we consider a voltage source as always associated with a series impedance and a current source as associated with a shunt impedance. Network branches based on these constraints and allowable in CIRCAL-1 are shown in Fig. 3.3. First, this approach makes it unnecessary to include junctions which are not electrical nodes in the data structure. These arise whenever a voltage source is inserted in a branch.

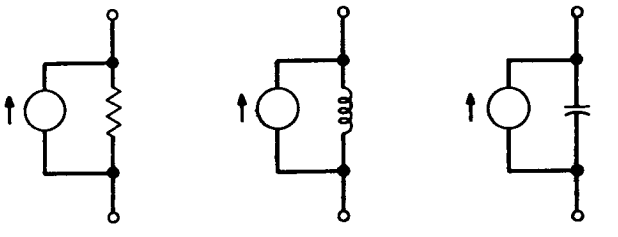
Secondly, it reduces all branches with linear elements to one basic "canonic form", hence (for this preliminary version) eliminating the need for an identifier and reducing system complexity. It is clear that this convention imposes no topological restrictions on the circuits that can be handled, since any element connected across a voltage source or in series with a current source may be removed, and connections of the type shown in Fig. 3.4(a) can always be changed into their equivalent forms, shown in Fig. 3.4(b).

2. Formation of the Data Structure

The data structure for a network is composed of three basic building blocks: There are blocks of storage registers or "beads" representing the electrical nodes in the network. Herein are stored the node potentials (E), the sum of the currents flowing into the node (DI) and other pertinent information shown in greater detail in Fig. 3.5(a). Other beads called parameter lists or "P lists" which contain most of the essential information about the branches are shown in Fig. 3.5(b). The P lists are connected in strings to the nodes through intermediate beads called "junction boxes". There are generally two junction boxes for each P list, and each junction box has a register or component to indicate the polarity of the source in the branch (+1 if the positive terminal of the source is toward the node to which the junction box is connected, -1 otherwise). In addition, each junction box indicates the node to which the other end of the element is connected. The last junction box in each string references a location in memory called the "tie point". All actual referencing and interconnecting of beads is accomplished by providing a register or part of a register in a given bead called a "pointer" whose value is the address of the first item in the referenced bead. The statements "A is a pointer to B" or "A points to B" mean that the contents of A is the address of B. Fig. 3.5(b) shows how a branch consisting of a sinusoidal voltage source and series-inductor is mapped into the P-list.

All beads in the data structure except the nodes are referenced by pointers in some other bead. Pointers to the nodes are stored in an array.





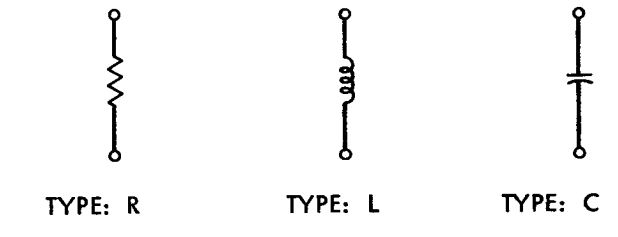
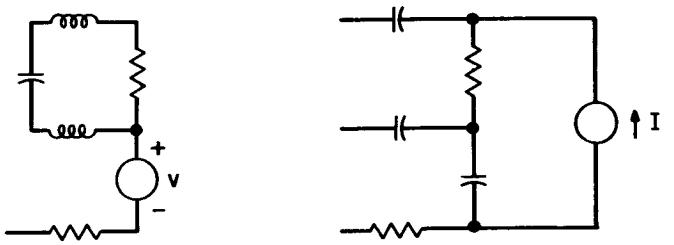
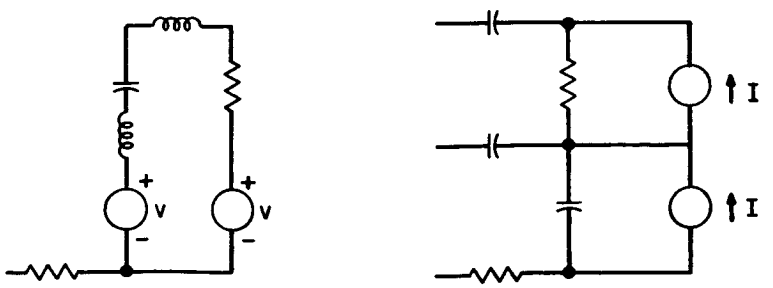


Fig. 3.3 Allowable Forms for Network Branches

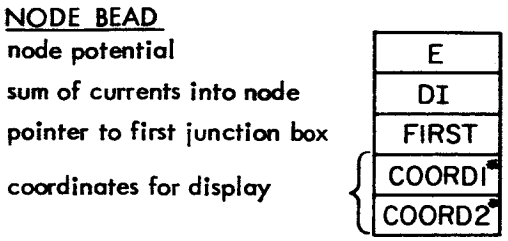


(a) Source connections not allowed

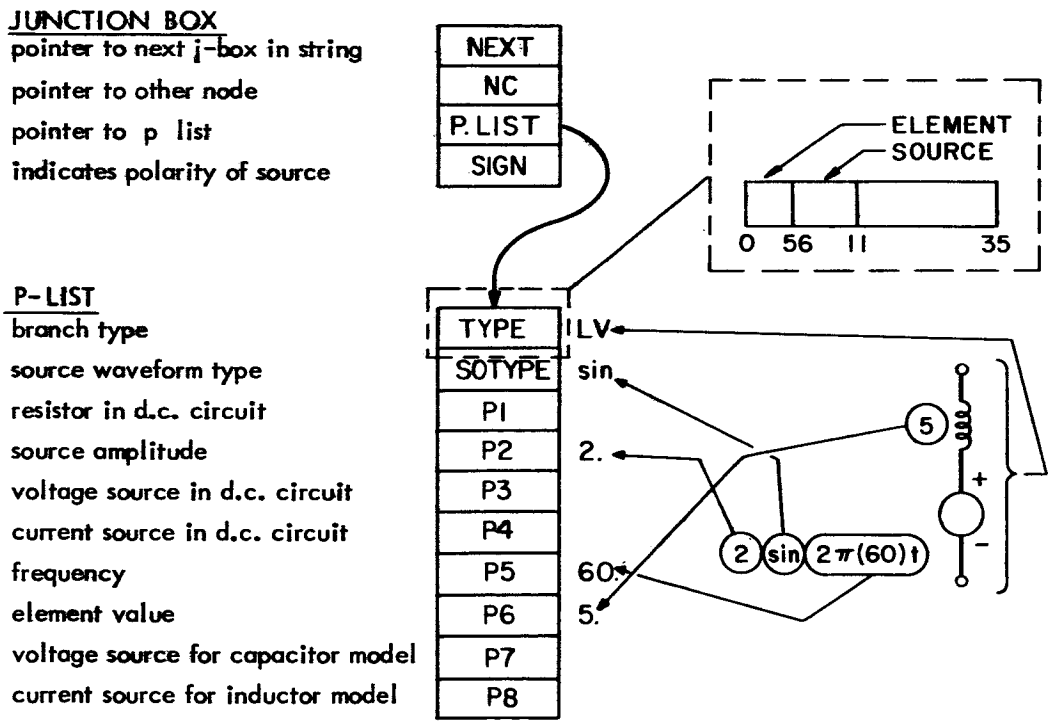


(b) Equivalent acceptable forms

Fig. 3.4 Conversion of Networks to Acceptable Form



(a) Node Bead



(b) Junction Box and P List

Fig. 3.5 Blocks of Storage or "Beads" Comprising the Data Structure

In order to clarify data structure organization, consider the circuit of Fig. 3.6. Fig. 3.7 shows a step-by-step development of the data structure for this circuit as the circuit is fed into the computer. The nodes are numbered in progressive order with the datum or zero voltage node as zero, while P-lists and junction boxes are automatically introduced for each element. It is assumed

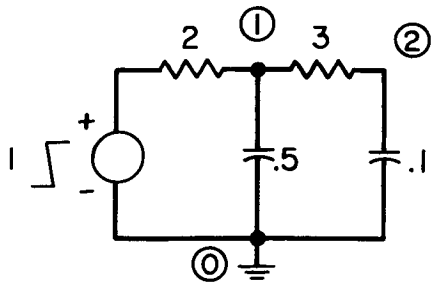


Fig. 3.6 Typical Circuit to be Analyzed

here that the typer alone is used as an input device, although the same approach holds when the CRT is used. The statements which are successively typed to feed the network into the computer are shown on the top of Figs. 3.7(a), (b), (c) and (d). The organization of these statements is explained next.

The subroutine TOPO which generates the data structure accepts statements of the form:

N1, N2, T1, V1, V2, T2, V3

where

N1: 1st node (positive terminal of the source is toward N1)

N2: 2nd node

T1: branch TYPE: RV, LV, CV, RI, LI, CI, R, L, C as shown in Fig. 3.3

V1: element value (ohms, henrys, farads)

V2: source amplitude (volts, amperes)

T2: source waveform: sin, cos, u (for unit step)

V3: source frequency (cps) (necessary only if T2 is sin or cos.)

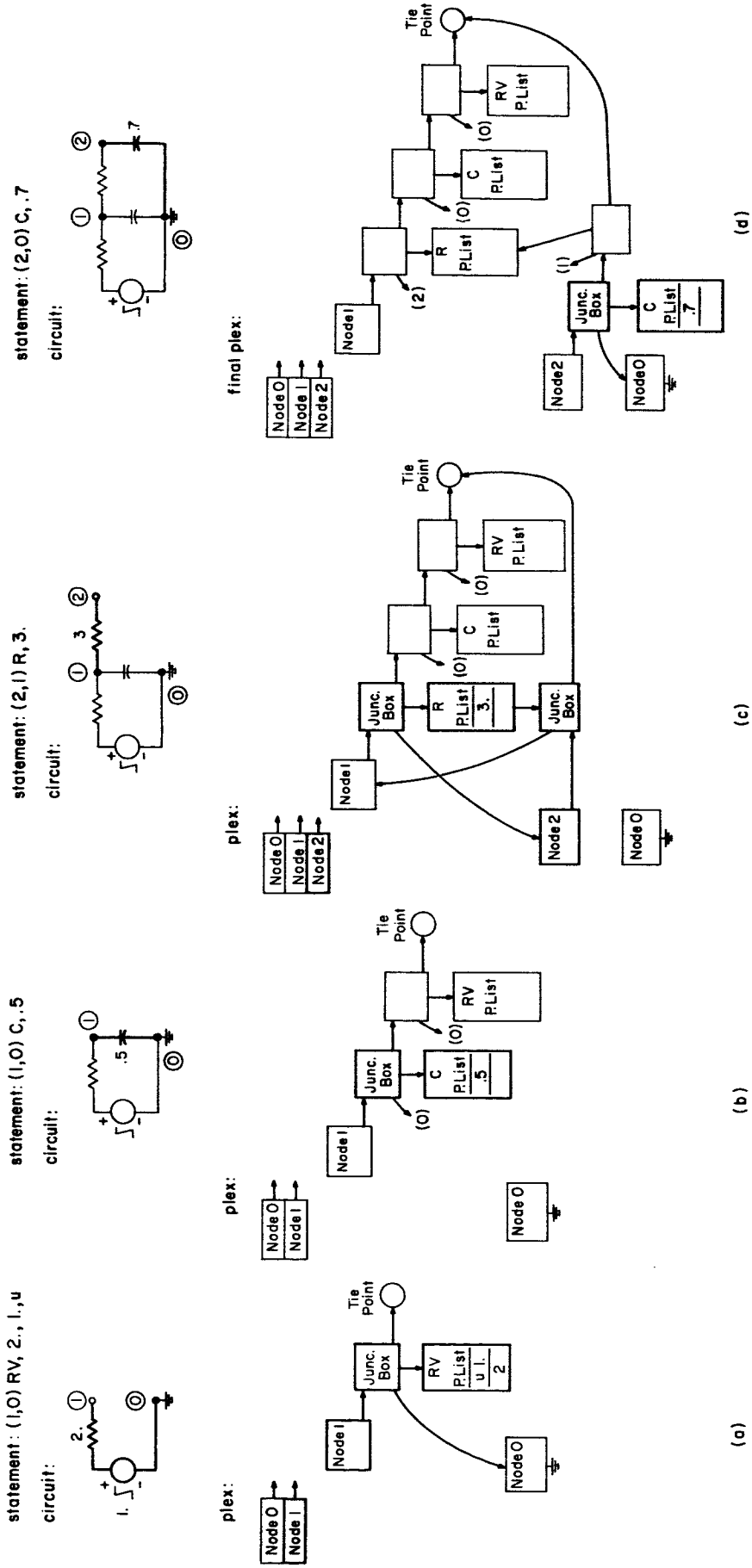


Fig. 3.7 Progressive Development of Data Structure for the Circuit of Fig. 3.6

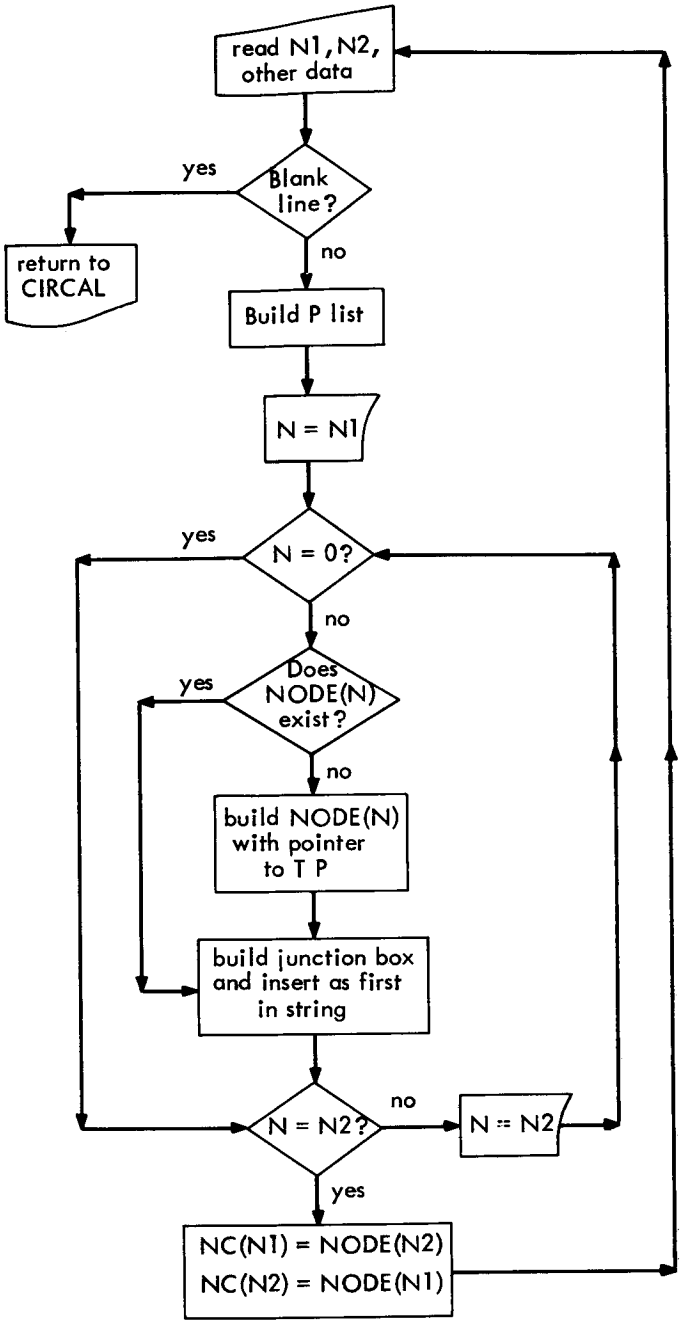


Fig. 3.8 Flow Chart of the Subprogram TOPO which Generates the Data Structure

Upon admission of such data, the program generates a P list and proceeds to determine if node N1 already exists. If it does, then the branch consisting of a junction box pointing to the P list is inserted as the first item in the string of junction boxes. The component NEXT of the newly defined junction box is set pointing to the junction box which was previously pointed to by the node. The component FIRST of the node, is now set pointing to the new junction box. If the node does not already exist, a new node is created and the junction box with P list is inserted as the only item in its string. The foregoing action is repeated for node N2. If in either case the node is the zero node (which exists a priori), then no action is taken to initiate a string. After the above process has been repeated for both nodes, the component NC in each junction box is set pointing to the opposite node. A flow chart for the data structure generation is shown in Fig. 3.8. Detailed flow charts and program listings are contained in Appendix D.

3. Conversion of the Dynamic Problem to a Set of Static Problems

Subroutine DYSOL performs the operations linking the dynamic and the amnesic problems. The subprogram MNIP which produces the amnesic solution is to be provided with a network the branches of which contain at most one linear resistor, one constant current source and one constant voltage source connected as in Fig. 3.9. Parameters P1, P3, and P4 of the P list are recognized by MNIP as the

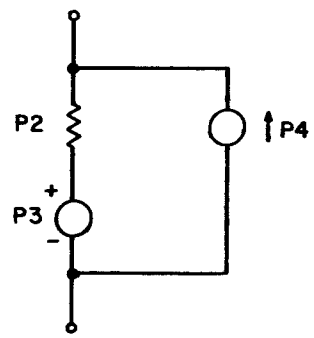


Fig. 3.9 Canonic Form for Network Branches

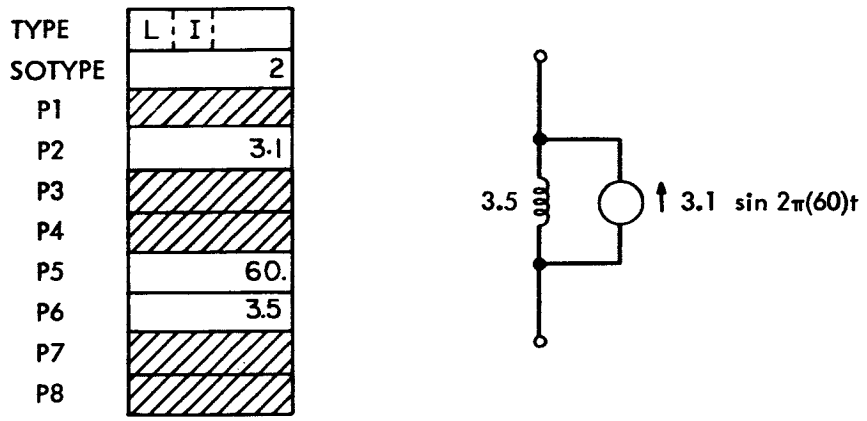
resistor, voltage source, and current source respectively. The function of DYSOL then, is to convert the dynamic problem into a sequence of static problems, by transforming all branch elements into canonic form.

To illustrate, consider a branch containing an inductor and a parallel current source. Fig. 3.10 demonstrates the use of various P-list components during the conversion. Parameters referring to the original branch description have been previously filled in by Subroutine TOPO. Subroutine DYSOL converts the inductor to its equivalent static form, (Fig. 3.10(b)), computing the values of the resistor (P1) and equivalent source (P8) from Eqs. 2.34 and 2.35, respectively. The value of the source at the particular time of consideration is computed and temporarily stored in a variable called SAMPLE. In the final step, SAMPLE is added to P8 to form P4, and P3 is set to zero. The resulting canonic form is depicted in Fig. 3.10(c). A similar procedure is carried out for the remaining branch types shown in Fig. 3.3. The flow chart of Fig. 3.11 traces the steps followed by the subroutine in the conversion of these branches. The foregoing operations are performed on each branch, by the double loop shown in Fig. 3.12. Here X is a pointer to the particular junction box being considered. It is originally set to quantity FIRST of the node under consideration, so that it references the first item in the string of junction boxes.

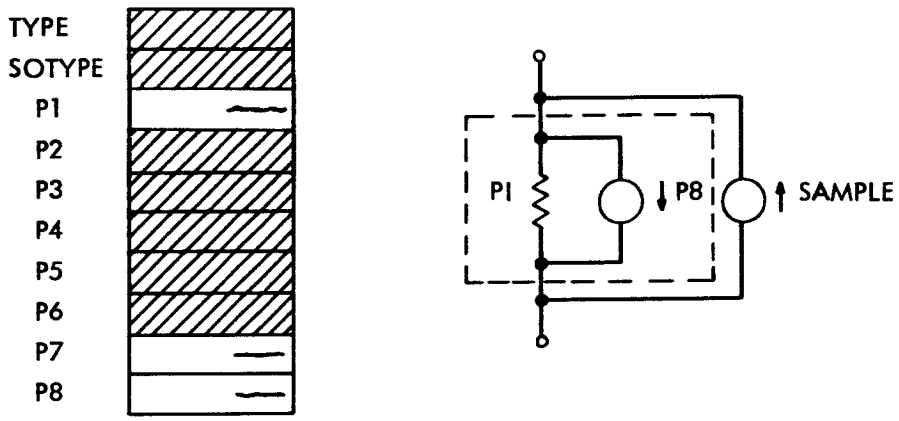
After operating on a branch, X is set to NEXT of the present junction box, thus pointing to the next junction box in the string (see Figs. 3.5 and 3.7 for further clarification). Each time, X is checked to see if it points to TP. If not, the program begins operation on the P list associated with the junction box to which X is pointing. If on the other hand pointer X does point to TP, the operations are resumed for the next node. Notice from Fig. 3.7 that if a branch is connected between two nodes neither of which is the ground node, it will be referenced by two junction boxes. In order to prevent performing the conversion operation on the same P list twice, conversion is performed only if the sign of the junction box is +1. The remaining portions of the chart are self explanatory. When all branches have been converted, control is transferred to Subroutine MNIP which computes the solution of the resulting amnesic network.

4. Solution of the Amnesic Problem

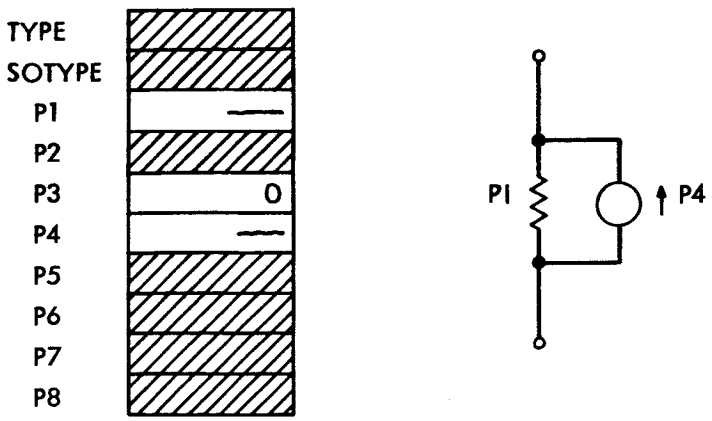
This section shows how the relaxation method for the solution of linear amnesic networks described in Section B of Chapter II is



(a) P List as set up by TOPO



(b) Intermediate Form



(c) Final Form used by MNIP

Fig. 3.10 Conversion of Network Branch to Canonic Form

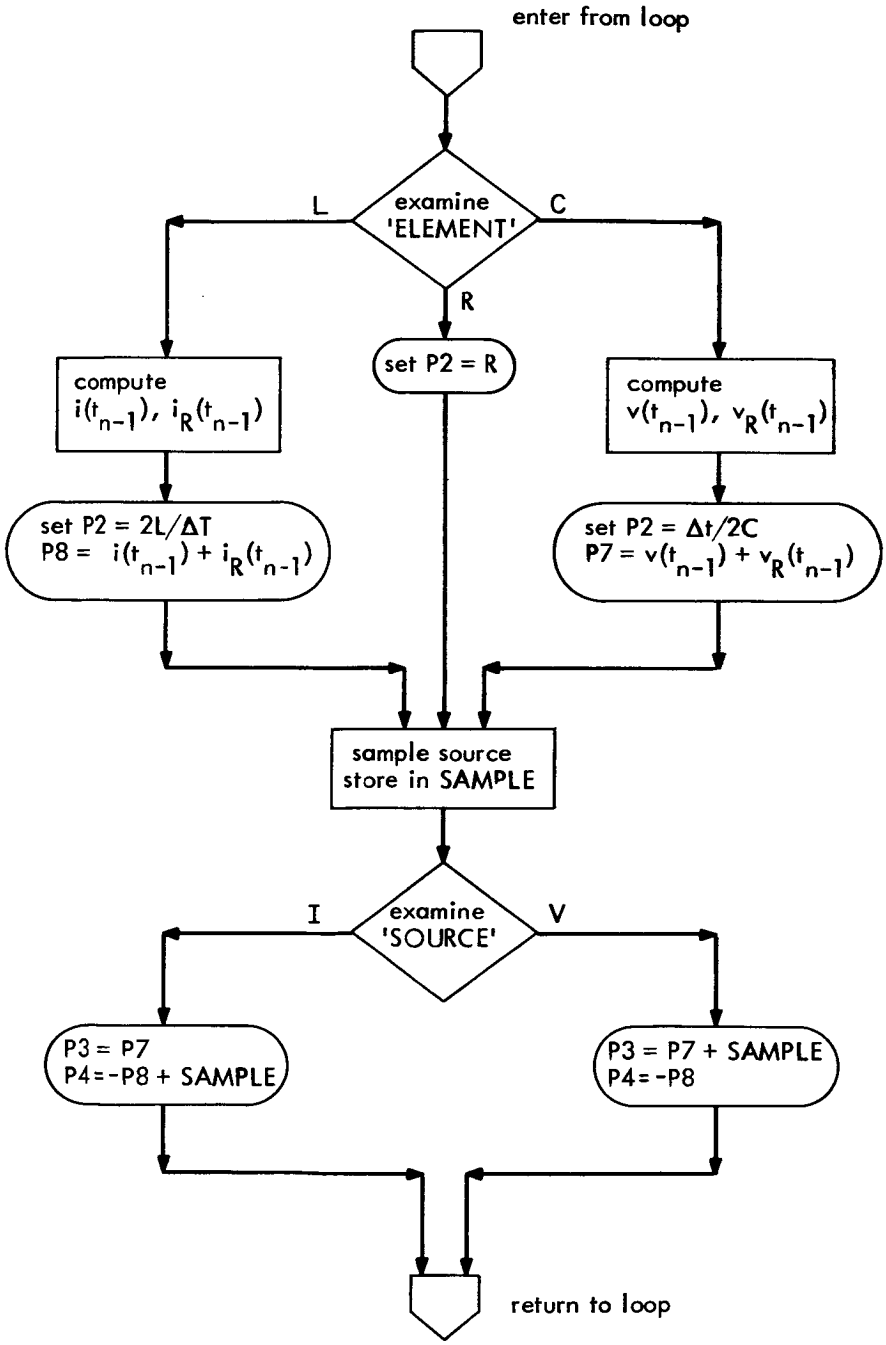


Fig. 3.11 Flow Chart for Branch Conversion (DYSOL Subprogram)

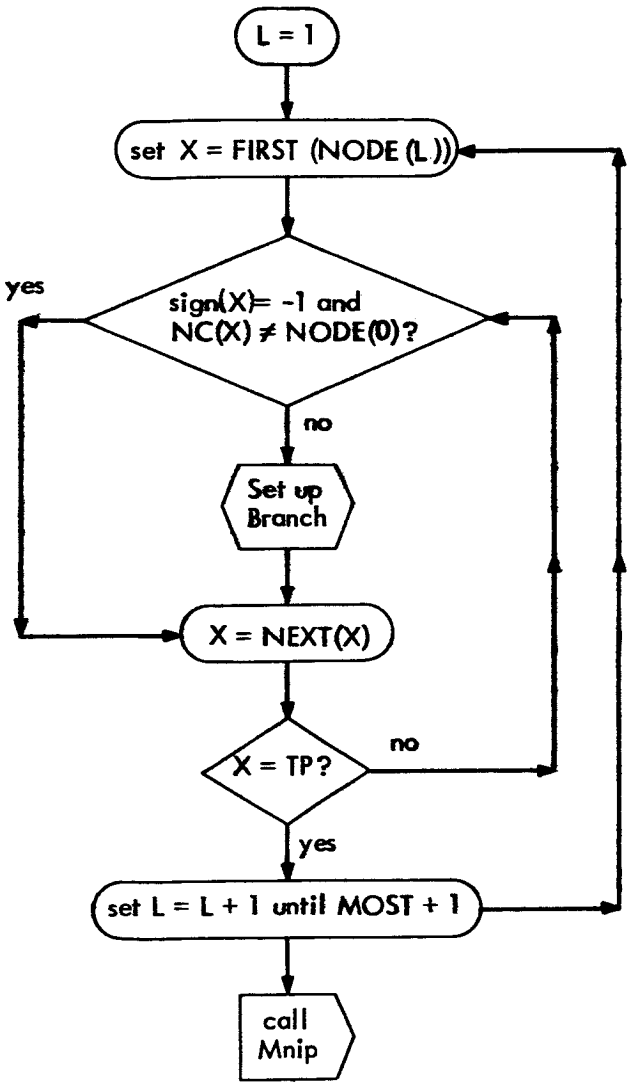


Fig. 3.12 Looping Mechanism used in DYSOL

implemented. Subroutine MNIP (for Modified Newton Iteration Procedure) performs this function on the amnesic network generated by DYSOL.

Recall from Chapter II that the relaxation method for solution of the network involves assumption of a set of node potentials $\{e_j^m\}$, computation of the sum of the currents at each node, and finally modification of each node potential by adding to it a constant times the sum of the currents into that node. The procedure is iterated until the current error becomes tolerably small. We may think of the MNIP program as containing an operator which travels along the string of branches connected to a node, computes the currents in the branches, sums these currents at the node, and stores the sum, $\Delta I(L)$, where L is a node index in the node bead. The basic operations are shown in the flow chart of Fig. 3.13. When $\Delta I(L)$ is computed for all nodes, the current error is compared with the given tolerance δ. If it exceeds δ, then new node potentials are computed by Eq. 2.3 and the entire process is repeated until a set $\{e_j^p\}$ yielding a current error less than δ is reached.

It was taken for granted in the above discussion, that a suitable value for the convergence constant K was available. Let us see how such a value might be obtained. Recall from Chapter II that the change, U(K), of the current error is a convex upward parabola with maximum at $K = \frac{K_I}{2}$ where K_I is the least upper bound on values of K yielding convergence. Our approach is the following. Two points K_1 and K_2 are selected, such that

$$\text{a) } K_1 > K_I$$

$$\text{b) } \frac{1}{2} K_I < K_2 < K_I$$

The value of U(K) at K_1 and K_2 is computed and a straight line is passed through these points as illustrated in Fig. 3.14. The point where that line crosses the K axis is denoted by K'_2 and the ordinate $U(K'_2)$ is computed. A straight line is then passed through $(K_2, U(K_2))$ and $(K'_2, U(K'_2))$ to determine a new intersection $K'_1 > K_I$. The foregoing process is repeated recursively using K'_1 and K'_2 in

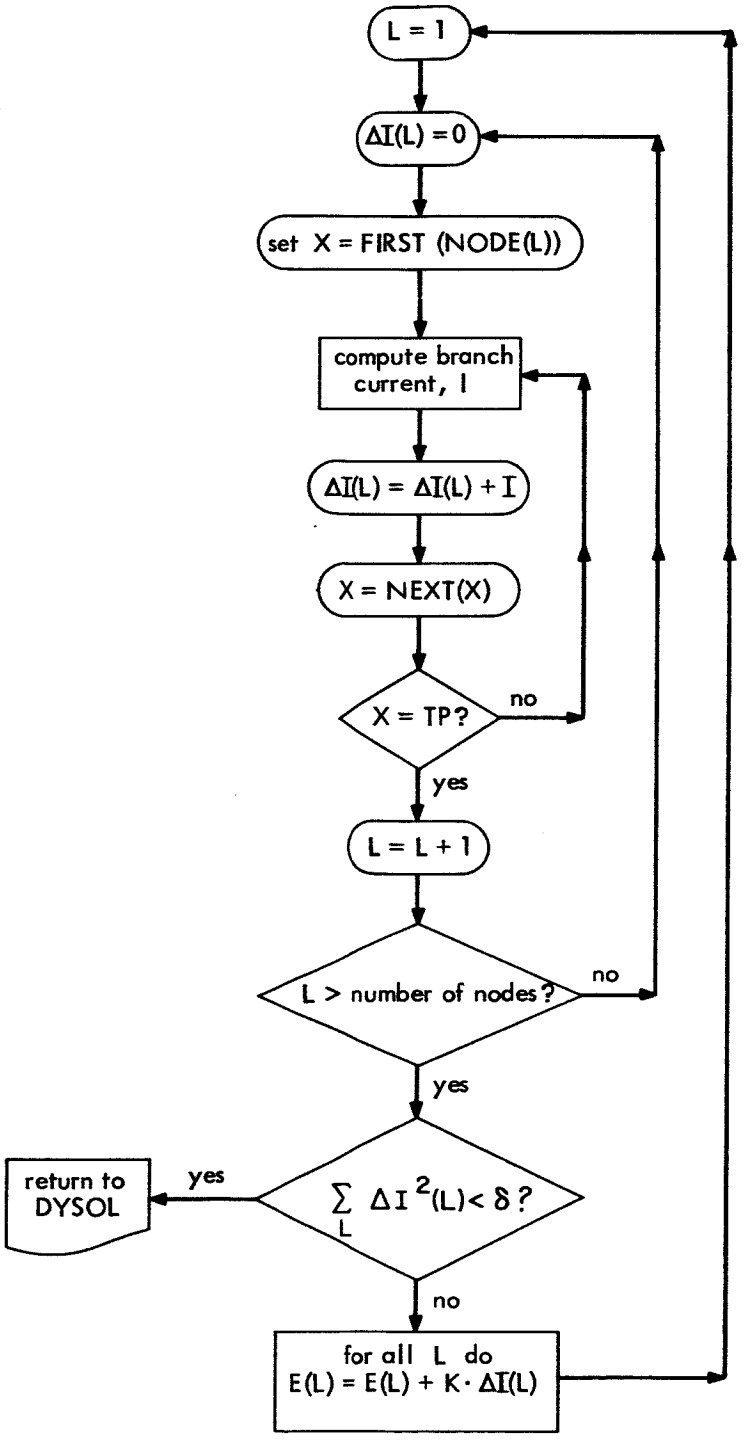


Fig. 3.13 Application of Perturbation Method to Data Structure (MNIP Subroutine)

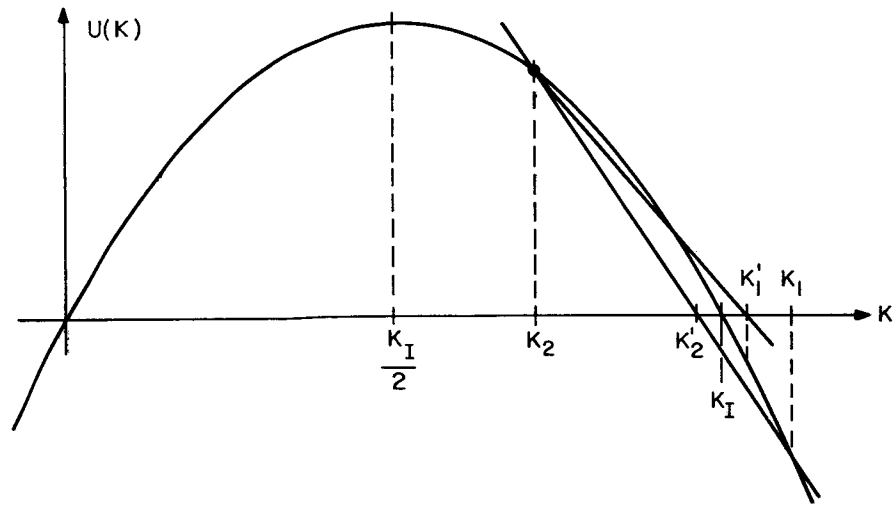


Fig. 3.14 Estimating the Lowest Upper Bound, K_I, on the Convergence Curve

place of K_1 and K_2 until the difference of K_1' and K_2' is less than some small tolerance, e.g., 1 Percent of the average estimate, $\frac{1}{2}(K_1' + K_2')$; that is it is required that

$$\frac{2(K_1' - K_2')}{(K'_1 + K'_2)} < 0.01 \qquad (3.1)$$

When 3.1 is satisfied, a value of K equal to half the computed upper limit is chosen, i.e.,

$$K_{opt} \cong \frac{1}{2}[\frac{1}{2}(K_1' + K_2')] \qquad (3.2)$$

It can be shown from geometric considerations that the least upper bound K_I can be found with any desired accuracy if initial points are chosen satisfying conditions a) and b) above. In order to initially obtain two points K_1 and K_2 satisfying these conditions, the following binary-search procedure is employed. A value, K_x, of K, is chosen and it is determined, by examining the sign of $U(K_x)$, whether K_x is greater or less than K_I. If K_x is less than K_I, then K_x is doubled to obtain the new point $2K_x$. If the ordinate at $2K_x$ is positive indicating that $2K_x$ is also less than K_I, then $2K_x$ is taken as the original point and the above procedure is repeated until a point greater than K_I is found. A similar procedure is employed if the originally chosen point, K_x, is greater than K_I. In this case successively smaller values are computed by halving K_x.

It has been found experimentally (see Chapter IV) that the optimum value of K for the entire iteration process is "fairly close" to the optimum value for one iteration. Hence, the search for the optimum value of K is performed only once at the first call of the subroutine MNIP. Thereafter, optimum K is recomputed only if the particular value of K currently being used does not reduce the current error in successive iterations. A flow-chart depicting the operations of Fig. 3.14 is shown in Fig. 3.15.

5. The Output Program

The output program provides means for examining the voltage across pairs of nodes in the network.

The voltage as a function of time may be listed in numerical form, plotted on the typewriter, or displayed on the cathode ray tube. All three forms of output are controlled by the subroutine of .PRNT. This subprogram searches out the voltages to be printed or plotted, and selects the appropriate output device. A flow diagram for .PRNT is shown in Fig. 3.16.

To use .PRNT , a command is given to CIRCAL, of the form

COMMAND V(N1, N2) MB MA

where N1 and N2 are the nodes across which the voltage is to be read, MB is the number of time increments to be printed starting at the MAth computed point, and COMMAND determines the output mode (typewriter listing, typewriter plotting or CRT display). The underlined quantities are transmitted via the argument list to .PRNT. The subprogram examines MB and if it is zero, sets MB equal to the total number N of solution points that have been computed. The program then checks to see if the last point to be plotted (the MB + MAth) is greater than the total number N of points computed. If it is, then MB is set to N-MA so that only the computed points will be plotted. The program then selects the appropriate time points from array T and stores them in array TS. It also computes the difference between the potentials at node N1 and N2 and stores these values in array F. The program then examines the COMMAND and transfers to the appropriate output device.

D. EXAMPLE OF OPERATION

The following presents a record of a typing session with CIRCAL-1 where it is desired to evaluate the step response of a series LC circuit.

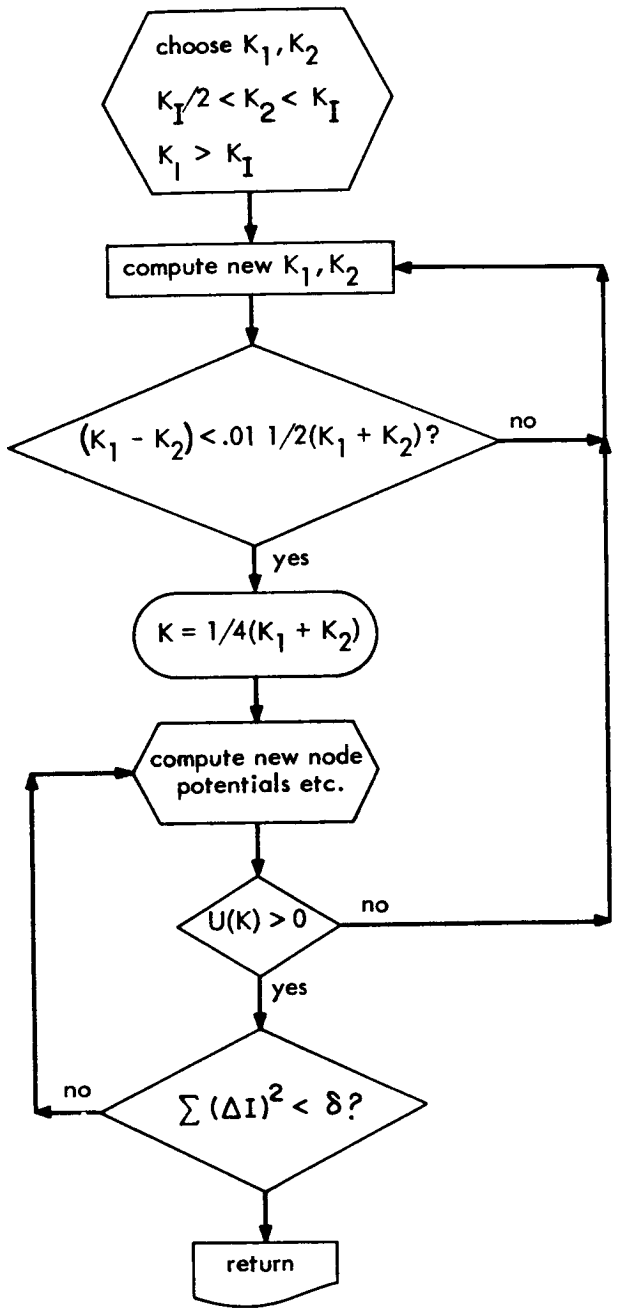


Fig. 3.15 Estimating the Optimum Value of K (MNIP Subroutine)

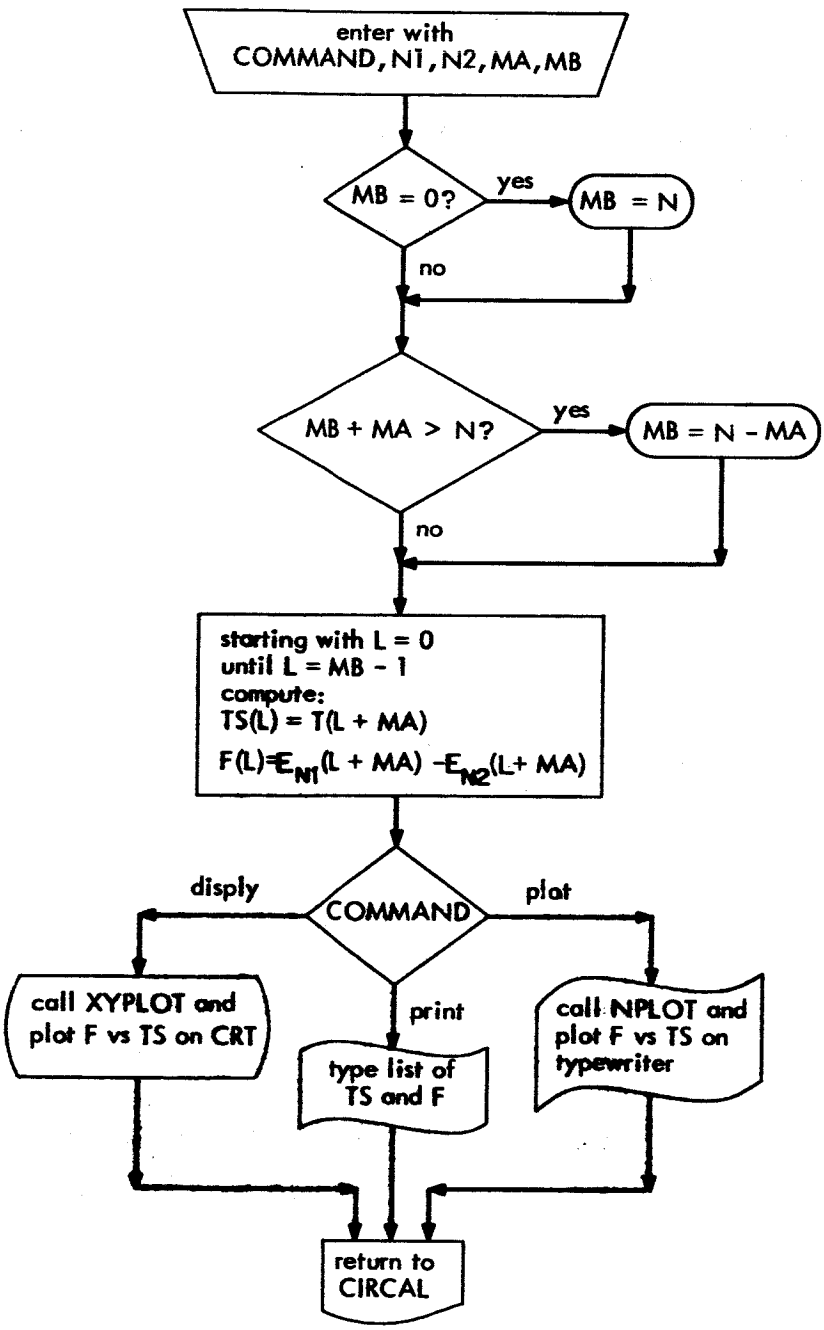


Fig. 3.16 Flow Chart for .PRNT

Machine response is typed in capital letters and user commands are in lower case letters. An initial carriage return is given to obtain a list of the commands. Command "input" is then given and the branch descriptions are typed according to the format described in section C(2). A final carriage return ends the input sequence. Command "analiz" is then given. A time increment of .2 seconds is chosen and 35 points of the solution are requested. The voltage from node 1 to datum is then listed for 10 time increments and plotted on the typewriter for 35 increments by giving the commands "print" and "plot" respectively. Naturally, a different time increment, display mode or number of plotted points could have been requested in the session to suit the users desire. Observe that the time taken to solve this problem is 2.3 seconds.

```
EXECUTION.

COMM.

THE COMMANDS ARE

INPUT
ANALIZ
PRINT F(N1,N2) MB MA
PLOT F(N1,N2) MB MA
DISPLY F(N1,N2) MB MA
ERASE
QUIT


COMM. input

TYPE BRANCH DESCRIPTION.
 1 0    lv      1.      1.     u
 1 0     c      1.

COMM. analiz

TYPE TIME INCREMENT, NO. OF POINTS, ONE ITEM PER LINE.
.2
 35

COMM. print v(1,0) 10
```

LIST OF V(1, 0)

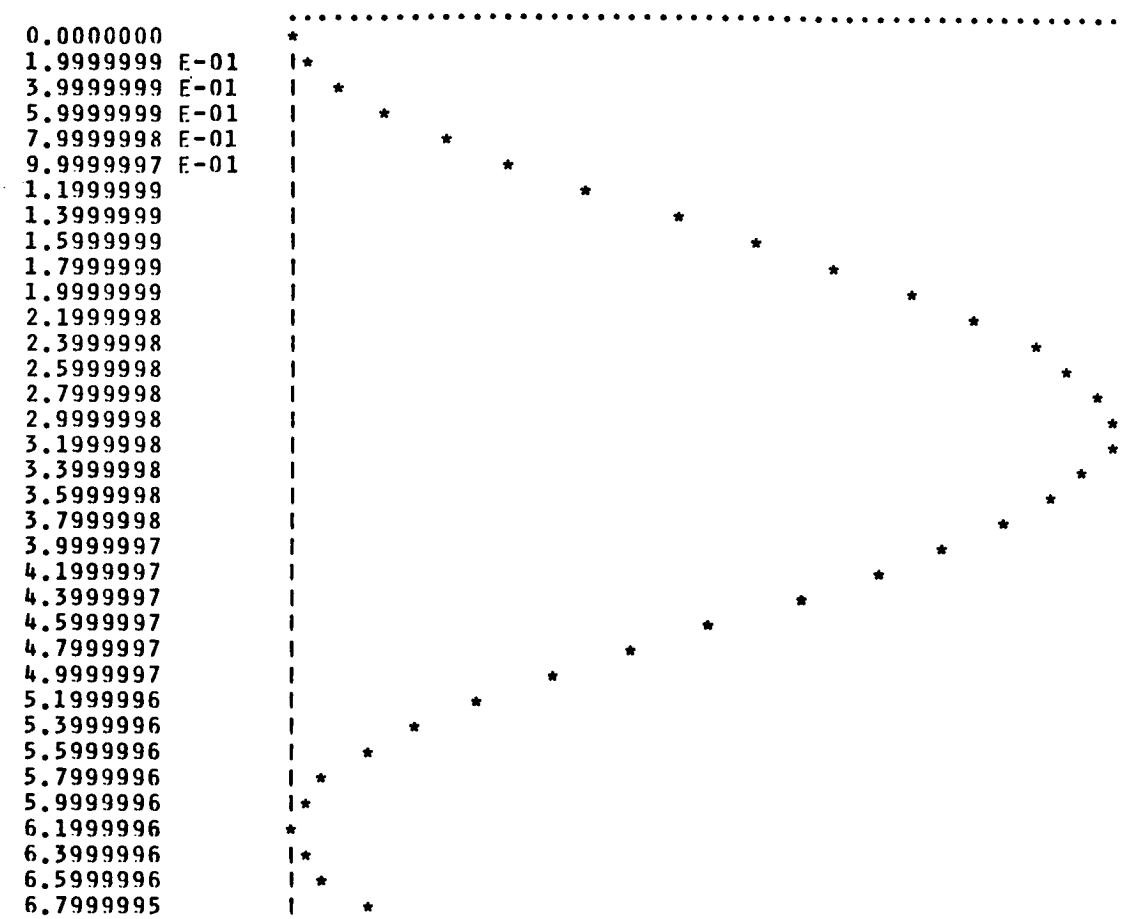
TIME	FUNCTION	TIME	FUNCTION
0.	9.8638799E-03	2.0000000E-01	4.8893244E-02
4.0000000E-01	1.2533970E-01	5.9999999E-01	2.3657144E-01
7.9999999E-01	3.7832359E-01	9.9999998E-01	5.4469581E-01
1.2000000E 00	7.2909928E-01	1.4000000E 00	9.2423109E-01
1.5999999E 00	1.1223634E 00	1.7999999E 00	1.3156498E 00

```
COMM. plot v(1,0)
```

GRAPH OF V(1 ,0)

SCALE IS 3.76211151 E-02 PER SPACE

TIME IS LISTED BELOW.



COMM. quit

R 2.616+8.600

CHAPTER IV

RESULTS AND ANALYSIS

A. INTRODUCTION

This chapter demonstrates the use of CIRCAL-1 in the solution of some simple circuits. In addition, the results of some programs which were composed early in the development are examined in order to evaluate relevant errors, convergence rates, solution time, and other pertinent data.

We shall examine first the results of a special program written to investigate the properties of the iterative method of Chapter II and we shall compare some of this data with the theoretical considerations of that chapter. We shall then pass on to the more general circuit analysis program of Chapter III (CIRCAL-1) and examine convergence of the amnesic solution. Two numerical integration methods other than the one used in CIRCAL-1 will be described. The errors generated by these techniques will be compared to the errors generated using the trapezoidal method of Chapter II. Some examples of networks analyzed using the present version of CIRCAL will be presented. Finally, some "experimental" results will be presented on the number of iterations required for solution of the amnesic problem.

B. ORIGINAL INVESTIGATIONS

In order to obtain some idea of the feasibility of the iteration technique presented in Chapter II, a preliminary program was written to apply this method to the network of Fig. 4.1. It was desired to determine:

1) Which values of the convergence constant K yield a converging solution for this network and which values bring about most rapid convergence?

2) In what manner does the solution converge (linearly, exponentially, etc.) and approximately how many iterations are required for any specified tolerance?

3) How is the total number of iterations affected by the initial values of the node potentials, $\{e_j^o\}$?

4) How are the foregoing results modified by insertion of a non-linear branch in the network?

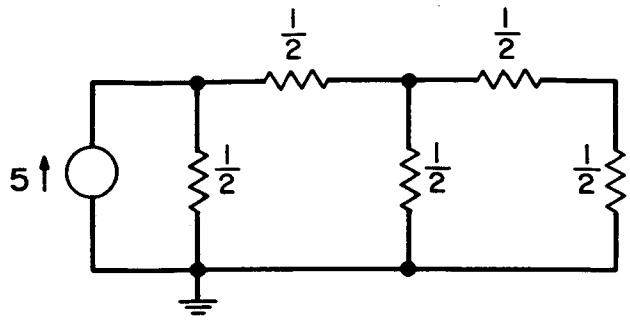


Fig. 4.1 Network used for Preliminary Investigation of the Perturbation Method

A number of computer runs were made using different values of K. Table 4.1 lists the results of a sequence of such runs. Here, the current-error tolerance, δ, was set at 0.1, and the solution was computed starting with all of the node potentials at zero. Values of the convergence constant K, larger than 0.250 did not yield convergence. At K = 0.250 and K = .005 the solution was converging very slowly (more than 80 iterations had not brought the solution close to the equilibrium value). The value of K yielding most rapid convergence (8 iterations) was 0.20. Since the upper limit K_I for this case is somewhere near 0.25, it might seem to the reader that the optimum value of K should have occured somewhere near K = 0.125. Recall, however, from Chapter II that the optimum value of K at one iteration is not necessarily optimal for succeeding iterations.

Computer runs were also made to investigate convergence for several values of the current error tolerance. It was found that for values of K around 0.2, the current error could be reduced to a value as low as 10^{-7}, without requiring a change in K. This slow variation of the optimum value of K for any iteration has been observed in a number of examples.

Table 4.1

Number of Iterations Versus Convergence Constant

Conv. Constant K	No. of Iterations
0.250	∞
0.245	75
0.200	8
0.150	11
0.100	17
0.050	33
0.005	∞

Current error $\delta = 0.10$

Initial values $e_1 = e_2 = e_3 = 0.00$

It has given rise to the strategy used in the final program, that is actually computing a value of K which is optimum for the first iteration and using this value for all successive iterations as long as the current error is decreasing. It can also be noted in this example that starting with node potentials further away from the solution does not substantially increase the number of iterations required for convergence, a phenomenon which has been substantiated theoretically in Chapter II.

The convergence of the node potentials of the network in Fig. 4.1 from initial values of zero to their equilibrium values is illustrated in Fig. 4.2. In this example the value of K was originally set to 1.0 and decreased by increments of 0.1 whenever reduction of the current error was not realized. Observe that no convergence occurs until K reaches a value of 0.2 which yields convergence for the remaining iterations. The current error is plotted on this graph for K = .2 and is repeated in Fig. 4.3 on a logarithmic scale versus the number of iterations plotted on a linear scale.

The straight line with negative slope of Fig. 4.3 indicates that the current error decreases exponentially with increasing iterations.

Fig. 4.4 displays the logarithm of the current error versus the number of iterations for various values of the convergence constant. It can be seen from that figure that the slope of the line, which represents the rate of convergence, is steepest at K = 0.2 and is

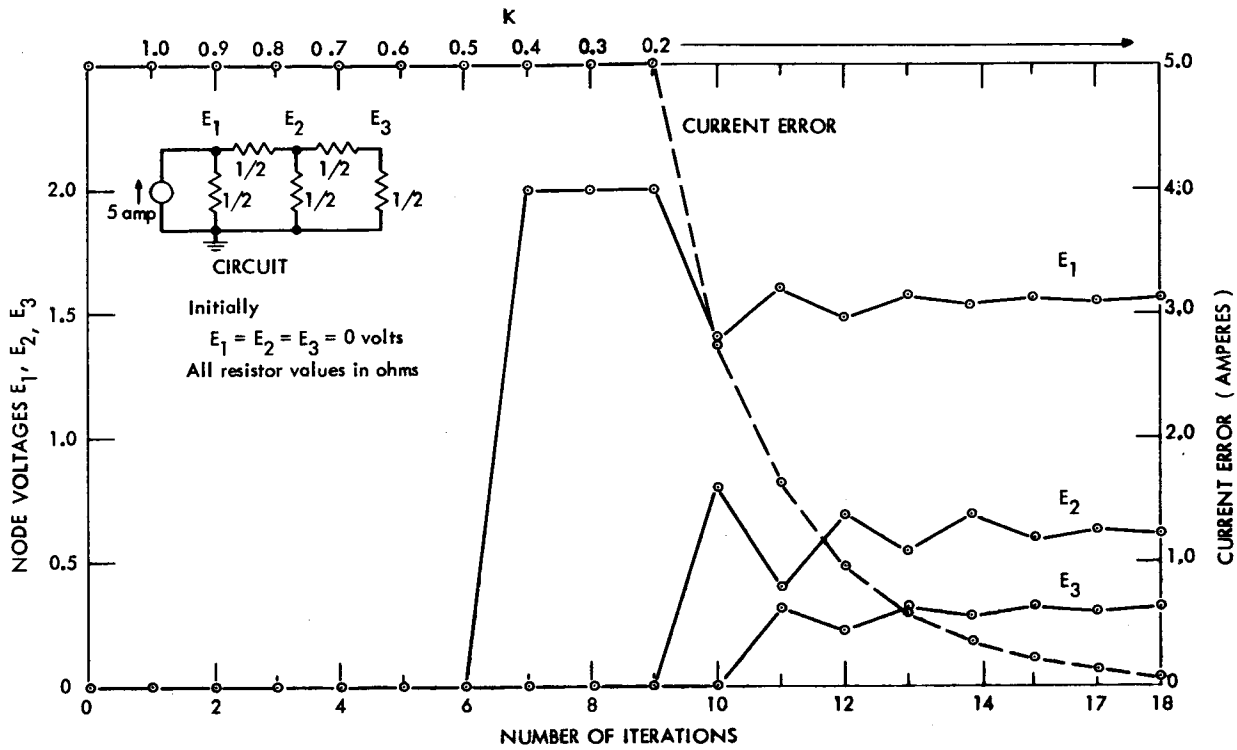


Fig. 4.2 Current Error and Node Voltages vs. Iteration Number and K

less steep for values of K both greater and less than 0.2. For K = .005 (near the lower limit) and K = 0.245 (near the upper limit) convergence is still exponential, but very slow. Observe that the rate of convergence is not exactly exponential in Fig. 4.4 although the lower bound for that rate was shown to be an exponential, if optimum values of K are used at each iteration.

A nonlinear element with the i-v characteristics shown in Fig. 4.5(b) was inserted next in one of the network branches as shown in Fig. 4.5(a). The scale factor C was adjusted so that the equilibrium node potentials would be the same as for the linear case. Table 4.2 compares the number of iterations required for this

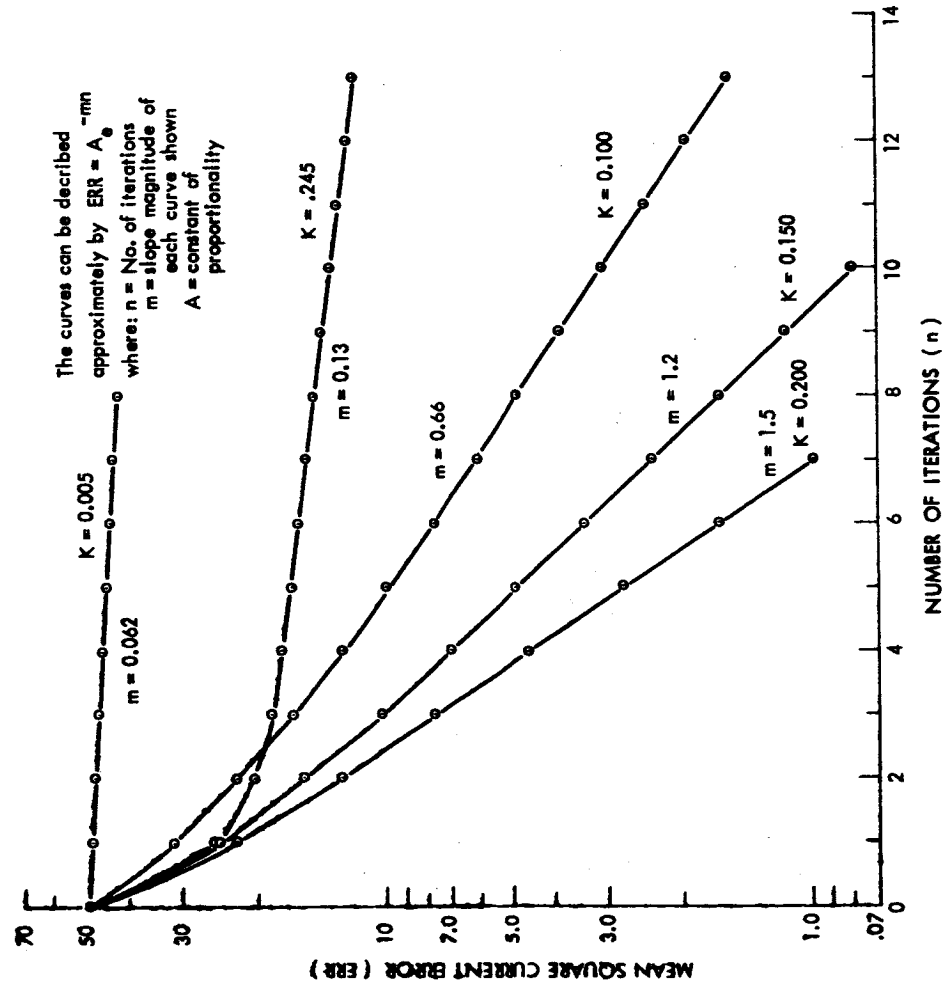


Fig. 4.4 Change in Current Error vs. Number of Iterations for Various Values of the Convergence Constant K

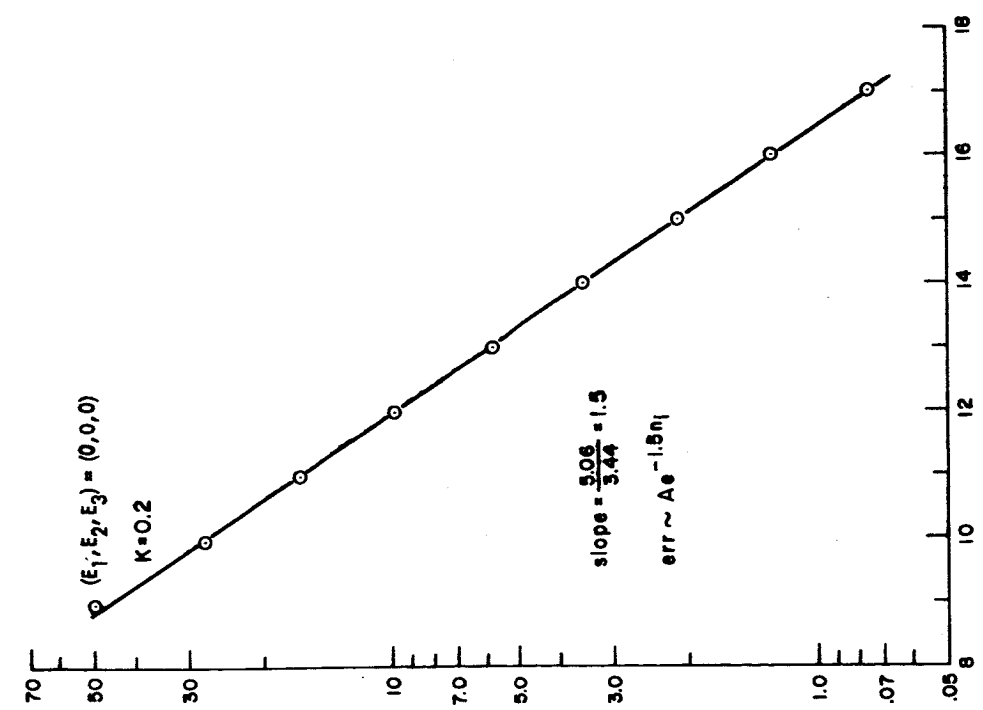


Fig. 4.3 Change in Current Error vs. Number of Iterations for the Network of Fig. 4.1

nonlinear network to the number required for the linear network of Fig. 4.1 for some values of the current error and starting node potentials. Although more iterations seem to be required to reach equilibrium in the nonlinear case, this increase is certainly not great. Some further experiments were carried out with both this network

Table 4.2

Number of Iterations Required for Solution of Linear and Nonlinear Networks

Initial Values (e_1, e_2, e_3)	Current Error	No. of Iterations	
		Linear Network	With Nonlinear Branch
1, 1, 1	0.1	9	11
0, 0, 0	0.1	8	9
-1,-1,-1	0.1	10	10
0, 0, 0	.001	16	21

Convergence constant K = 0.2

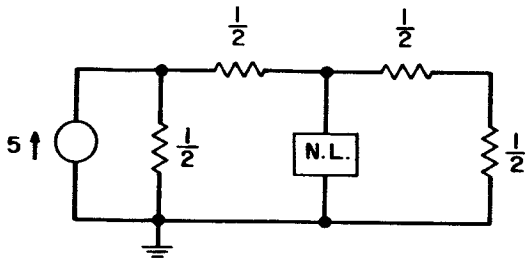
and a similar network using a nonlinear device with terminal characteristics $v = Ci^3$. In all cases convergence rate was found to be similar to the convergence rate of the linear network.

Table 4.3

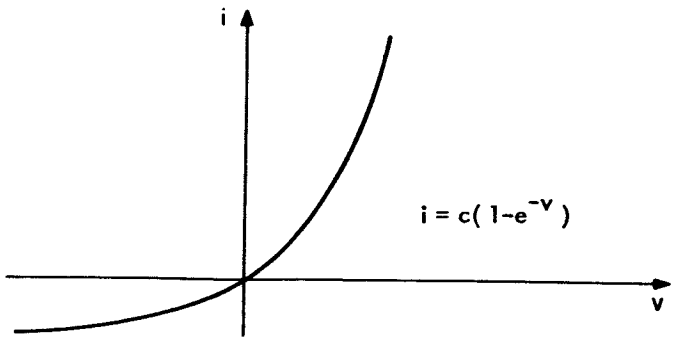
Maximum of Convergence Curve for Values of the Voltage Source

Value of Source, V	Height at peak, $U(K_I/2)$
1.0	.04
2.0	.16
3.0	.36
0.5	.01

$\Delta t = 0.5$



(a) Insertion of nonlinear element in the network



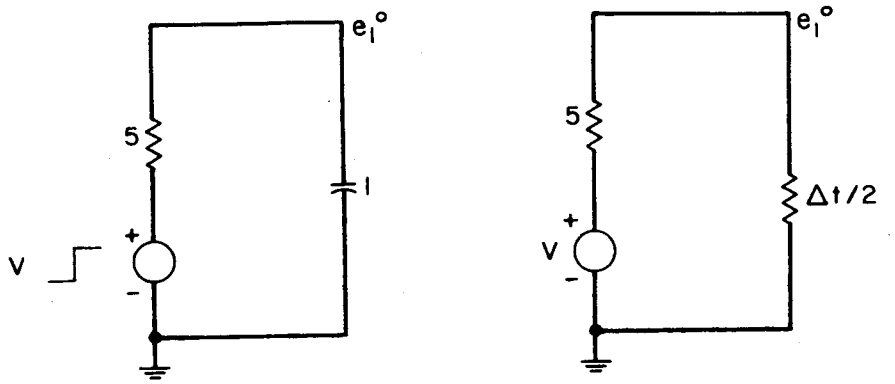
(b) i-v Characteristics of nonlinear element

Fig. 4.5 Linear Network with Nonlinear Branch

C. INVESTIGATIONS OF THE GENERAL CIRCUIT ANALYSIS PROGRAM

1. Further Considerations of Convergence

This section verifies theoretical results concerning V(K) and examines variations of V(K) with circuit parameters. The network of Fig. 4.6(a) was used in this experiment. Analysis was carried



(a) R C network (b) Equivalent amnesic network for first time instant

Fig. 4.6 RC Network used for Investigation of Convergence in the Nonamnesic Case

out for the first increment of time. In this time increment the capacitor is represented as shown in Fig. 4.6(b) by a resistor of value $\frac{\Delta t}{2C}$ and no source. The equilibrium equation for this amnesic network is

$$E_1\left(\frac{2}{\Delta t} + \frac{1}{5}\right) = \frac{V}{5} \tag{4.1}$$

The conductance matrix (a scalar in this case) is

$$G = \frac{2}{\Delta t} + \frac{1}{5} = \frac{10 + \Delta t}{50\Delta t} \tag{4.2}$$

Using Eq. 2.30 to evaluate the upper limit of the convergence constant for this network yields

$$K_I = 2\,\frac{\Delta I_1\, G\, \Delta I_1}{\Delta I_1\, G^2\, \Delta I_1} = \frac{2}{G} = \frac{10\Delta t}{10 + \Delta t} \tag{4.3}$$

In this simple one-node case, the upper limit K_I is independent of

the current error, hence of the original node voltages, and of the independent source in the network. Similar results will be obtained in the case of a larger network if its conductance matrix is diagonally dominant. For $\Delta t = 0.5$ Eq. 4.3 becomes

$$K_I = .476$$

The actual convergence curves obtained by choosing node potentials, and applying the relaxation method for particular values of K is shown in Fig. 4.7. For $\Delta t = 0.5$, Fig. 4.7 shows that the upper limit K_I is the same as that calculated above. Eq. 4.3 shows

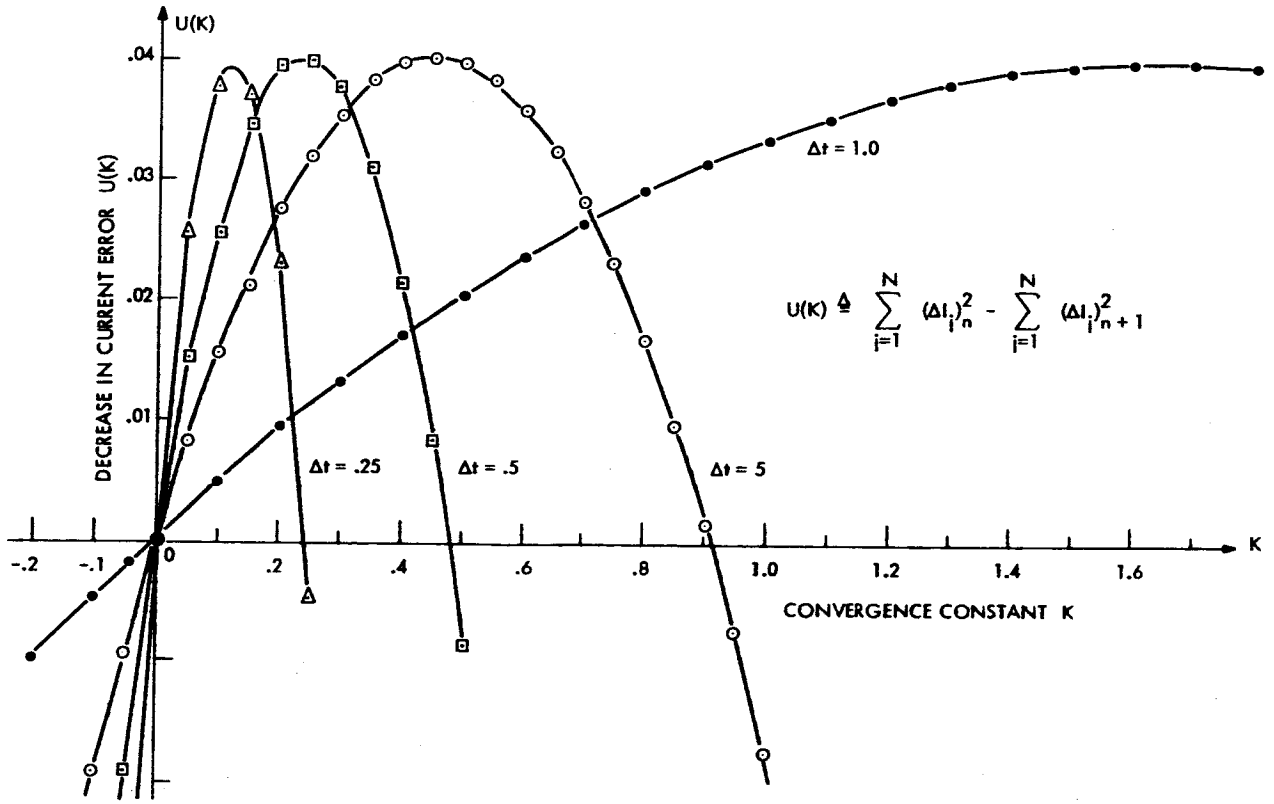


Fig. 4.7 Convergence Curves for the Network of Fig. 4.6

approximate linear dependence of K on Δt for small values of Δt. The convergence curves of Fig. 4.7 exhibit the same dependence. The height of the curves is independent of Δt. However, the height appears to be proportional to the square of the voltage

source as shown in Table 4.3. The results of Chapter II verify this observation as shown in the following. From Fig. 2.1 the maximum value of U(K) for the above network is

$$U(K)\Big|_{max} = U(\frac{K_I}{2}) = \frac{(\Delta I_1\ G\ \Delta I_1)^2}{\Delta I_1\ G^2\ \Delta I_1} \tag{4.4}$$

$$= (\Delta I_1)^2$$

From Fig. 4.6(b) we can write

$$\Delta I_1 = \frac{V}{5} - e_1^{\,o}\,(\frac{1}{5} + \frac{2}{\Delta t}) \tag{4.5}$$

Squaring this expression and substituting the result in Eq. 4.4 yields

$$U(K)\Big|_{max} = \frac{V^2}{25} - Ve_1^{\,o}\frac{2}{5}(\frac{1}{5} + \frac{2}{\Delta t}) + (e_1^{\,o})^2(\frac{1}{5} + \frac{2}{\Delta t})^2 \tag{4.6}$$

In the foregoing computation the initial node potential e_1^o was taken to be zero. Hence Eq. 4.6 reduces to

$$U(K)\Big|_{max} = \frac{V^2}{25} \tag{4.7}$$

which is independent of Δt and confirms the results of Table 4.3. At time instants when the voltage source in the capacitor is not zero, convergence is slower, since V in Eq. 4.7 is decreased by the voltage across the capacitor. It is not reasonable to generalize this result and to expect that the maximum value of U(k) is dependent on the square of the voltage sources of an arbitrary network. That the maximum value of the convergence curve does depend in some way on the current error, as indicated by Eq. 4.4, is more generally true and reasonable in light of the exponential properties of the convergence rate demonstrated in Section B of this chapter.

2. Integration Techniques other than Trapezoidal

In Chapter II, it was shown that a non-amnesic element could be represented over an interval of time, Δt, by an amnesic element

with an associated updatable state. It was seen for example that the i-v relation for an inductor can be expressed as

$$i(t_n) = i(t_{n-1}) + \frac{1}{L} \int_{t_{n-1}}^{t_n} v(\tau)\, d\tau \qquad (2.31)$$

and the integral in Eq. 2.31 can be approximated by a function of the voltage $v(\tau)$ evaluated only at the end points of the interval. In Chapter II, the integral was approximated by the average of the integrand evaluated at the two end points, multiplied by the length of the interval, Δt. This method of integration is known as a trapezoidal rule or a modified Euler method.[15,25] Two other ways by which an integral can be approximated were programmed. These methods will be described, and compared qualitatively to the modified Euler method. The reasons for selecting the modified Euler method over the other two will then become clear.

a. Euler Method (Rectangular Integration). The integral of Eq. 2.31 can be approximated by the value of the integrand at one of the two end points multiplied by the time increment. Under this approximation, using the lower limit, Eq. 2.31 becomes

$$i(t_n) = i(t_{n-1}) + \frac{\Delta t}{L} v(t_{n-1}) \qquad (4.8)$$

or using the upper limit it becomes

$$i(t_n) = i(t_{n-1}) + \frac{\Delta t}{L} v(t_n) \qquad (4.9)$$

Eq. 4.8 is termed an open form,[15] since the value of the current at the nth instant, $i(t_n)$ is completely determined by the current and voltage at the n-1th instant and hence can be computed directly. Eq. 4.9 (the "closed form"), on the other hand, is more suited to our purposes since it specifies a relation between the voltage and current at the same instant of time, and hence can be physically represented by a circuit of the form shown in Fig. 2.4(a).

In both open and closed-form cases a staircase approximation is fitted to the curve $v(\tau)$ versus τ (see Fig. 4.8), and the area under

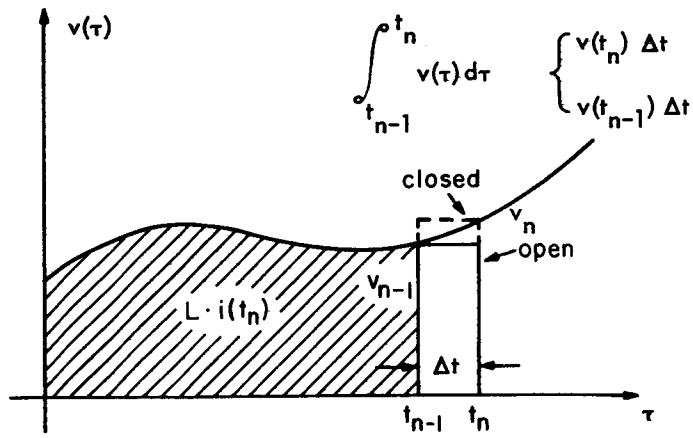
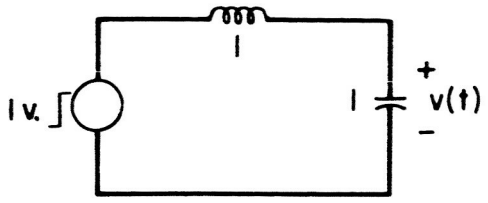


Fig. 4.8 Euler Methods of Integration

the curve is approximated by the area of this staircase function. It can be seen that in either case errors will accumulate more rapidly than in the modified Euler method if the slope of $v(\tau)$ does not change sign for some period of time. Clearly, if the slope remains positive, the open method will be undercompensating while the closed method will tend to be overcompensating.

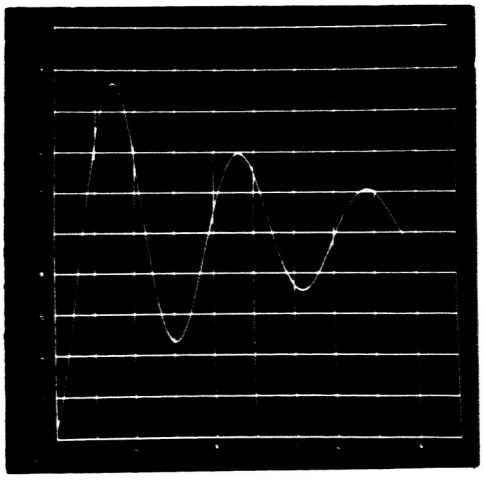
A program was written using the closed form to represent capacitors and inductors. The step response of a series L-C circuit computed by this integration technique is shown in Fig. 4.9(b) and is compared with the correct step response computed using the trapezoidal method of Chapter II. Instead of a pure sinusoid, an exponentially damped response is obtained in the case of the closed form. This damping can be easily explained[15] in terms of the numerical integration approximation used, but will not be included here. The application of the Euler method to some other lossy circuits such as a series RC was, however, successful in that the solution was computed with a small over-all error. A quantitative comparison of the errors obtained using this method to the errors obtained using the trapezoidal method, and the method to be presented next will be carried out in Part 3 of this section.

b. Simpson's Rule. In order to obtain a better approximation to the area under the curve $v(\tau)$ versus τ, a second-order curve is fitted through three points (two previously computed and the

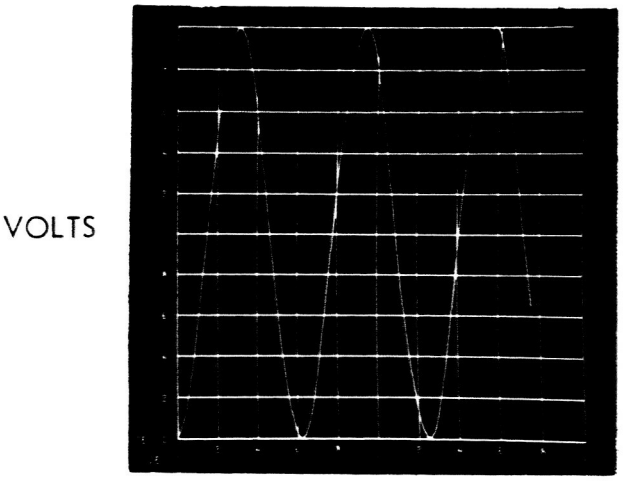


values in henrys and farads

(a) series L-C circuit



(b) voltage v(t) at capacitor computed using closed-form Euler method



(c) voltage v(t) computed with trapezoidal method

Fig. 4.9 Computation of Step Response for a Series LC Circuit using Closed-Form Euler and Trapezoidal Methods in Integration

present). This approach, known as Simpson's Rule, is illustrated in terms of an inductor as follows:

Eq. 2.31 may be rewritten as

$$i(t_n) = i(t_{n-2}) + \frac{1}{L} \int_{t_{n-2}}^{t_n} v(\tau)\, d\tau \tag{4.10}$$

If a second-order curve is passed through points $v(t_{n-2})$, $v(t_{n-1})$, and $v(t_n)$ the integral in Eq. 4.10 can be approximated by[25]

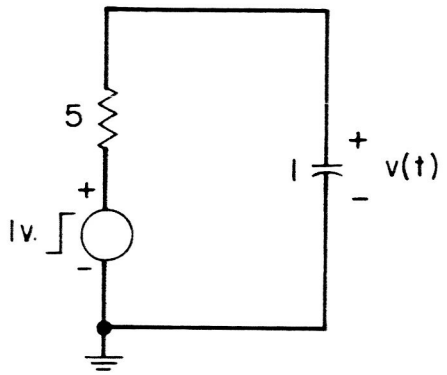
$$\int_{t_{n-2}}^{t_n} v(\tau)\ d\tau \approx \frac{1}{3}\ [v(t_{n-2}) + 4v(t_{n-1}) + v(t_n)]\Delta t \tag{4.11}$$

and hence Eq. 4.10 can be written as

$$i(t_n) = \left[i(t_{n-2}) + \frac{\Delta t}{3}\ v(t_{n-2}) + \frac{4\Delta t}{3} v(t_{n-1})\right] + \frac{\Delta t}{3}\ v(t_n) \tag{4.12}$$

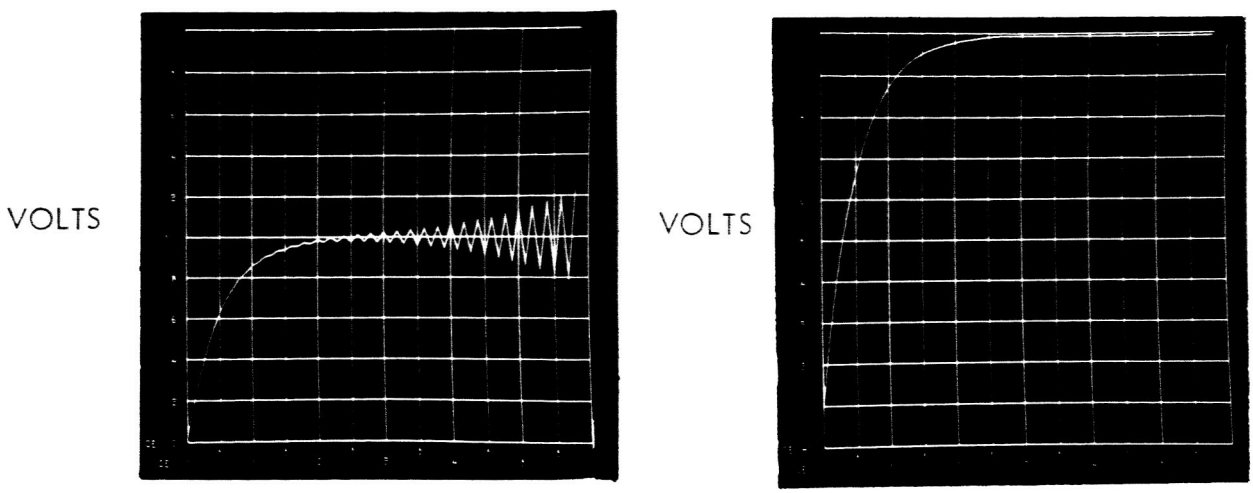
It is seen that this relation for the inductor can again be realized by a resistor and a current source in the form of Fig. 2.4(a), where $R_L = \frac{3}{\Delta t}$ and $I_n = i(t_{n-2}) + \frac{\Delta t}{3} v(t_{n-2}) + \frac{4\Delta t}{3}\ v(t_{n-1})$.

Although this second-order scheme is decidedly more accurate than either of the two methods previously described it can in many cases produce unstable solutions. This instability was experimentally observed when Simpson's Rule was applied to the RC circuit of Fig. 4.10(a). The step response of the circuit computed by Simpson's Rule and shown in Fig. 4.10(b) is compared with the correct solution computed by the trapezoidal method and shown in Fig. 4.10(c). Simpson's Rule seems to compute the response correctly for a while and then gives rise to exponentially growing oscillations about the correct solution. Hildebrand[15] shows that these oscillations called "parasitic solutions" will occur whenever any second or higher-order numerical method (such as that of Eq. 4.12) is applied to a system that can be described by a first-order differential equation. Besides the differential-equation eigenfunctions, this approach generates



values in ohms and farads

(a) R C Circuit



(b) v(t) computed from SIMPSON'S RULE

(c) v(t) computed using trapezoidal method

Fig. 4.10 Step Response of Series RC Circuit Computed using Simpson's Rule and Trapezoidal Method

eigenfunctions that increase in magnitude with the number of iterations. The exact form of these eigenfunctions for the network under consideration is derived in Appendix C and confirms the above experimental behavior.

Additional insight into the nature of the oscillations can be gained by considering the following simple example. The equation for an RC network is of the form

$$\dot{v}(t) = -\frac{1}{T} v(t) \tag{4.13}$$

where the dot denotes differentiation with respect to time. We shall consider here only the transient solution. Let us use a simplified second-order method of integration. In particular we shall use

$$v'(t_n) = v'(t_{n-2}) + 2\Delta t \ \dot{v}'(t_{n-1}) \tag{4.14}$$

where $v'(t_n)$ is the numerical approximation to $v(t_n)$. Fig. 4.11 shows what happens when Eqs. 4.13 and 4.14 are used to compute the solution. The true solution is shown dotted.

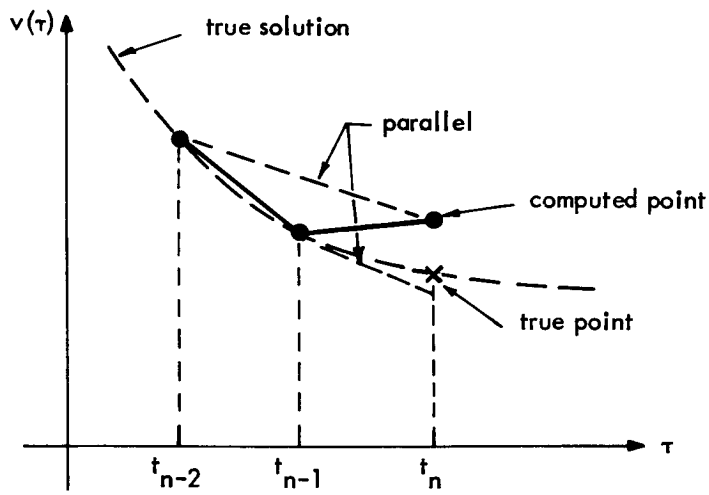


Fig. 4.11 Divergence of Second Order Integration Method

Assume that two adjacent points of the solution $v(t_{n-2})$ and $v(t_{n-1})$ have been computed exactly by some method. From the figure, it can be seen that $v(t_{n-1})$ is less than $v(t_{n-2})$, hence the slope at $\tau = t_{n-1}$ given by Eq. 4.13 will be less negative than

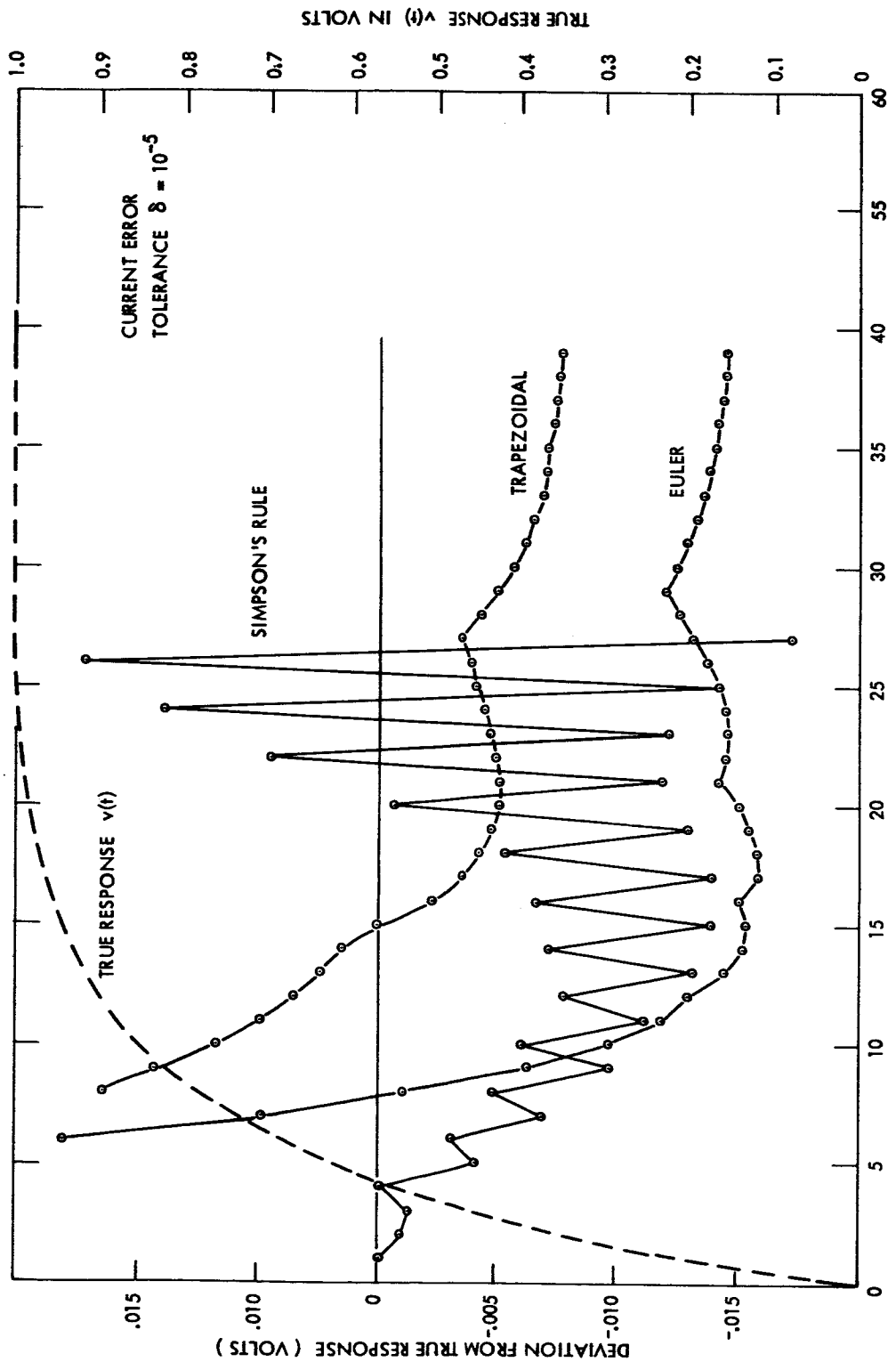


Fig. 4.12 Errors Generated in Computing the Step Response of the Series RC Network of Fig. 4.10

the slope at $\tau = t_{n-2}$. Since Eq. 4.14 uses this less negative slope to extrapolate a point $v'(t_n)$ starting at point $v(t_{n-2})$, it is possible for small Δt that $v'(t_n) > v(t_n)$. If, in addition, a numerical error causes $v'(t_n)$ to be greater than $v'(t_{n-1})$ (it can be shown that this will eventually be the case), then the next computed point $v'(t_{n+1})$ will appear below its correct value. From that point on, oscillations about the true solutions will increase exponentially.

3. Errors in Solution

In this section, errors generated in solving two simple networks by the foregoing numerical integration methods will be discussed. Theoretically, it is possible to estimate bounds on errors caused by the various numerical integration techniques, as shown in Appendix C. In order to gain further insight into the nature of these errors, this section examines experimental results obtained in the earlier phases of this research.

The step response of the series RC circuit of Fig. 4.10(a) was computed using each of the three integration methods described above. The exact solution for the voltage across the capacitor and the difference between that solution and the one obtained using each of the above three methods is plotted in Fig. 4.12. There is a comparatively large initial error (off scale) due to the representation of the capacitor over the first interval of time by a small resistor. This error can be reduced by using a smaller value for that resistor. Its effect can be, however, neglected for purposes of this analysis, since it introduces only a transient error. The error plot for Simpson's Rule is easily identified since it exhibits the undesirable parasitic oscillations that have been discussed in the foregoing. These oscillations increase steadily, and at a time of 22 seconds begin to alternate in sign, giving rise to the unstable response of Fig. 4.10(b).

The Euler method behaves well and yields errors of the order of 1.5 percent of full scale. That the final error is attributable to the current error tolerance chosen can be further verified by noting the change in the direction of the error at 29 seconds. At this time, the current error falls within the specified tolerance, δ, and hence the computed solution is held constant for the remaining time intervals. The exact solution, however, is still growing. Hence the error increases

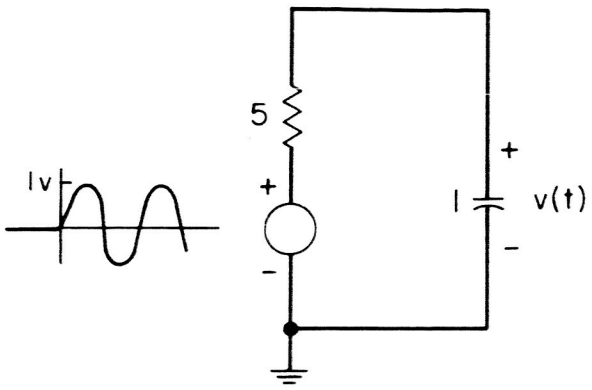
and asymptotically approaches a constant. Note that if the current error tolerance had been smaller, the computed voltage would not have saturated so early in time and the final error magnitude would have been smaller.

The error produced by the trapezoidal method is almost identical in its behavior to the error of the Euler method. However, the magnitude of the former is about half the magnitude of the latter. This is not surprising if it is recalled that the resistor used in the capacitor model is $\Delta t/2$ instead of Δt. This small resistor is significant in determining the change in node potential for a change of current. Hence for a given final current error the voltage error is reduced by a factor of 1/2.

In this example, the trapezoidal method seems to have no advantage over the Euler method except for an increase of solution accuracy. The accuracy produced by the trapezoidal method, however, can always be obtained with the Euler method by decreasing the current-error tolerance and allowing time for more iterations. However, it has been noted in Fig. 4.9 that the Euler method does not perform well in the analysis of the series LC network. The difference of the computed response and the exact solution of the network (shown in Fig. 4.9(a)) is plotted for the trapezoidal and Euler methods in Fig. 4.13. Besides being an order of magnitude larger, the error magnitude for the Euler method grows over one cycle. This is manifested in the solution as an exponential damping. Moreover, dilation of the period of the solution has occurred. The trapezoidal method exhibits neither a noticeable increase in the error nor a dilation of the period over the cycle. The solution of the network of Fig. 4.9(a) and the error produced by the trapezoidal technique are plotted for 6 1/2 cycles in Fig. 4.14. Although the maximum error over a cycle first decreases and then increases, by the end of 6 1/2 cycles the maximum error magnitude has not exceeded the maximum error for the first cycle.

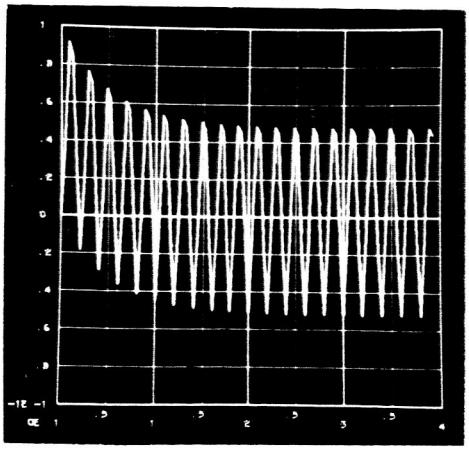
D. ADDITIONAL EXAMPLES

In this section four more examples of linear networks treated with CIRCAL-1 are given. Each of these was described to the



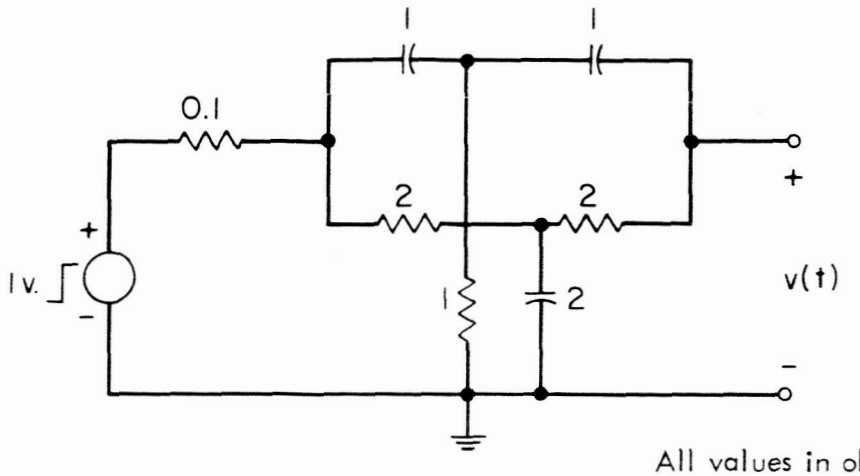
values in ohms and farads

(a) R C network with Sine Wave Excitation

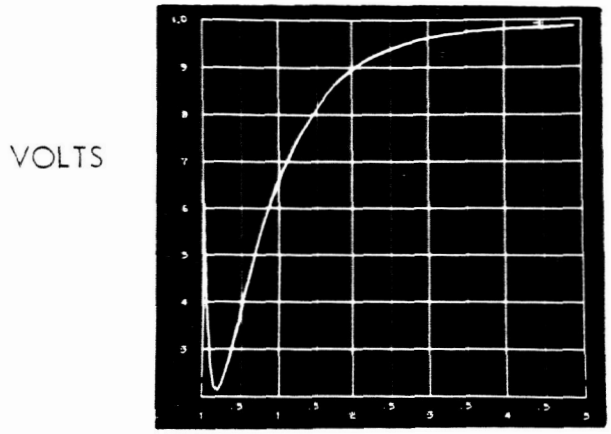


(b) Capacitor voltage v(t)

Fig. 4.15 Sinusoidal Response of RC Network



(a) Twin-T Filter



(b) voltage v(t) in Response to Unit Step

Fig. 4.16 Step Response of "Twin-T" Filter

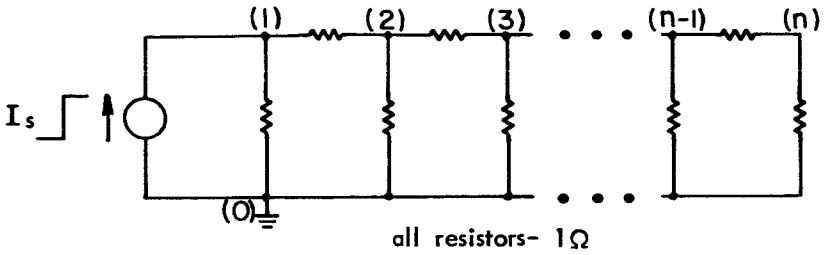


Fig. 4.19 Ladder Network used for Evaluating Number of Iterations as a Function of Network Size

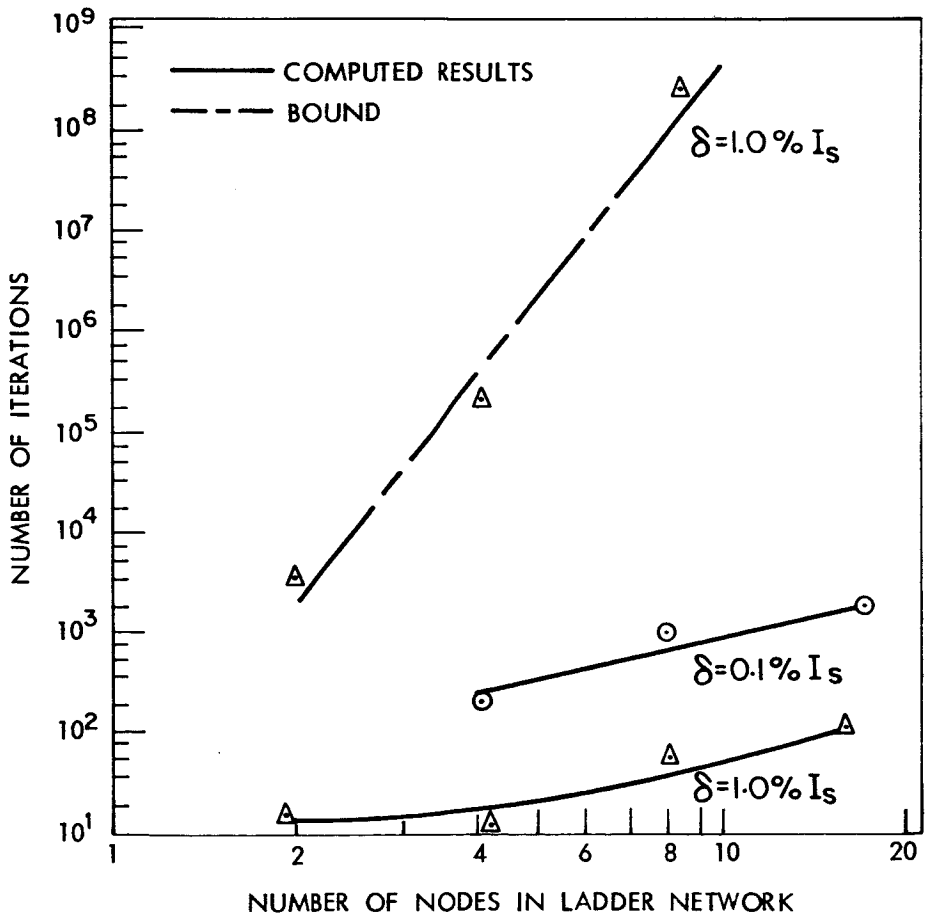


Fig. 4.20 Convergence Speed of Relaxation Method

of the number of nodes. The bounds derived in Chapter II are evaluated and plotted in Fig. 4.20 for the 1 percent case. These bounds are several orders of magnitude larger than the computed curves and exhibit dependence on n^8. The bounds are seen to be quite loose and of questionable value in estimating the number of iterations required for a given network.

CHAPTER V

SUMMARY AND CONCLUSIONS

A. SUMMARY

Evolution of CIRCAL-1, a computer program for the on-line simulation of electrical networks, is described. This version of CIRCAL (CIRCAL-1) is limited to linear networks with sinusoidal and step excitations. The relevant computational methods, however, are applicable to a larger class on nonlinear networks with arbitrary excitations. Users of the program compose with a light-pen the circuit to be analyzed on a cathode-ray tube (CRT) while typing component values on the teletype. Circuit description is in terms of conventional graphic symbols for basic circuit elements. As the circuit is composed on the CRT, a model of the network is formed within the computer. This model consists of computational building blocks which represent network branches and nodes in one-to-one correspondence with the actual network. The dynamic problem is then resolved into a number of static problems, solved in succession and integrated numerically to determine the state of the network at each instant of time. Each static problem is solved by relaxing the set of node potentials which were valid in the preceding time interval, in the presence of new independent source values, through use of Kirchoff's current law. Issues of convergence, stability and approximation are discussed from a theoretical and experimental viewpoint for the techniques used in CIRCAL-1, as well as for other techniques that were investigated. A number of examples are used throughout to motivate, illustrate and verify theoretical results.

B. CONCLUSIONS

CIRCAL-1 is the first version of a program aimed specifically at the simulation of electrical networks. The central objective here is the unrestricted on-line analysis and synthesis of electrical networks using a digital computer. More specific objectives were set forth in Chapter I and are summarized here.

1. Easy communication between designer and computer in conventional circuit terminology.

2. On-line editing of the network (modification of parameters, component values and topology) while results are being observed.

3. Provisions for use of special elements defined by the user (or the computer) in addition to standard circuit elements.

4. Hierarchy of compatibility (e.g. the ability to consider a large network as an element within a still larger network).

5. Special design aids such as simulation of environmental changes, aging of components, etc.

6. Growth capability, that is adaptability to future requirements which are not necessarily known at present.

CIRCAL-1, the first version of CIRCAL, was assembled with the foregoing objectives in mind. Not all of these objectives, however, were incorporated in this first version, since it was considered desirable to start with a simplified program that would evolve progressively toward more complex and more sophisticated versions. The benefit of such an approach is expedience and availability of "experimental" data early in the development. Such data, coupled to theoretical investigations, verifies concepts, reveals weak and strong points that could not be a priori anticipated, and often motivates future action.

For this reason, CIRCAL-1 was limited to analyze linear time-invariant, planar, resistor-inductor-capacitor, voltage-and current-source-excited networks where the excitations are either steps or sinusoids. These limitations apply only to the presently available working version of CIRCAL-1 within the program, the relevant computational structure has been made independent of these limitations. For example, the iterative procedure (MNIP) may be used to solve the static problem, at each state, for a larger class of nonlinear resistors. Alternatively, MNIP may be substituted with other static problem-solving methods. DYSOL, the portion of the program which sets up the dynamic problem in terms of many static problems and numerically integrates the results, is applicable to any network that can be physically realized, since it is based on the state-space representation of such a network. Likewise TOPO, the procedure responsible for setting up the data structure internal

to the computer is basically independent of all of the foregoing limitations, since it establishes a "mapping" of the network within the computer, regardless of the meaning or properties of the elements involved. The first of the foregoing objectives has been met through use of a cathode-ray-tube display and light pen. The designer communicates graphically with the computer in terms of conventional representations of circuit elements which he interconnects by manipulation of the light pen. This approach requires no special programming knowledge and can be explained to a typical designer in about five minutes. The specific implementations of this input-output approach is presented in a forthcoming report entitled "Graphical Communication for Electrical Network Simulation". Moreover, as was discussed in the foregoing, provisions have been made in CIRCAL-1 to complete the accomodation of objectives 2 through 6 in future versions.

CIRCAL-1 has demonstrated from a preliminary standpoint the overall feasibility of electrical-network simulation by use of a digital computer. More specific conclusions arising from use of CIRCAL-1 are as follows:

1. Although loose theoretical bounds on the number of iterations required to solve each static problem grow as the eighth power of the number of nodes, N, in the network, experimental results indicate a growth proportional to $N^{3/2}$. Matrix inversion techniques grow as N^3. Additional experimentation with both types of techniques is required, however, in order to increase the statistical sample and further confirm these results.

2. Trapezoidal integration seem quite adequate for future usage in updating the state of the network, since it combines acceptable accuracy with inherent stability.

3. Formation of the internal computer model corresponding to a given network is an adequate and rapid process. Few, if any, modifications of this process are anticipated for future versions.

4. Real-time editing of the network while observing results seem to be slightly handicapped by time delays inherent in time-sharing systems. These delays (presently ranging from a few seconds to a few minutes) prevent the simultaneous displaying of results and adjusting of network parameters. Instead, the man-machine combination behaves like a sampled-data system with a direct consequence that the editing process must be spread out over a longer time period.

APPENDIX A

PROOF OF THEOREM 2.1

Define N dimensional vectors by:

$$\underline{E} \triangleq \begin{bmatrix} E_1 \\ E_2 \\ \cdot \\ \cdot \\ \cdot \\ E_N \end{bmatrix} \qquad \underline{e}^m \triangleq \begin{bmatrix} e_1^m \\ e_2^m \\ \cdot \\ \cdot \\ \cdot \\ e_N^m \end{bmatrix} \qquad \underline{\Delta I} \triangleq \begin{bmatrix} \Delta I_1^m \\ \Delta I_2^m \\ \cdot \\ \cdot \\ \cdot \\ \Delta I_N^m \end{bmatrix} \tag{A.1}$$

Using this notation, Inequality 2.2 can be written as

$$(\underline{E}-\underline{e}^{m+1})^T(\underline{E}-\underline{e}^{m+1}) < (\underline{E}-\underline{e}^m)^T (\underline{E}-\underline{e}^m) \tag{A.2}$$

where the superscript T indicates transpose. Equation 2.3 becomes

$$\underline{e}^{m+1} = \underline{e}^m + K\,\underline{\Delta I}^m \tag{A.3}$$

It is desired to show that Inequality A.2 holds for suitable values of K in Equation A.3. If Equation A.3 is plugged into Equation A.2 the result is

$$[\underline{E}-(\underline{e}^m+K\,\underline{\Delta I}^m)]^T\,[\underline{E}-(\underline{e}^m+K\,\underline{\Delta I}^m)] \overset{?}{<} [\underline{E}-\underline{e}^m]^T[\underline{E}-\underline{e}^m] \tag{A.4}$$

(the question mark over the Inequality indicates that the statement is yet to be proven). Then if we regroup terms and expand we have

$$[\underline{E}-\underline{e}]^T[\underline{E}-\underline{e}] - K\,[\underline{E}-\underline{e}]^T\,\underline{\Delta I} - K\,\underline{\Delta I}^T\,[\underline{E}-\underline{e}]+K^2\underline{\Delta I}^T\,\underline{\Delta I} \overset{?}{<} [\underline{E}-\underline{e}]^T[\underline{E}-\underline{e}] \tag{A.5}$$

where we have dropped the superscript m for convenience.

If we cancel the common term, and note the equivalence of the second and third terms on the left hand sides of Inequality A.5 we obtain

$$-2K\,[\underline{E}-\underline{e}]^T\,\underline{\Delta I} + K^2\,\underline{\Delta I}^T\,\underline{\Delta I} \overset{?}{<} 0 \tag{A.6}$$

or the equivalent form

$$2K\ [\underline{E}-\underline{e}]^T\ \underline{\Delta I} - K^2\ \underline{\Delta I}^T\ \underline{\Delta I}\ \overset{?}{>}\ 0 \qquad (A.7)$$

Let us assume first that K is positive. Then if we divide Inequality A.7 by K we obtain the condition

$$2[\underline{E}-\underline{e}]^T\ \underline{\Delta I} - K\ \underline{\Delta I}^T\ \underline{\Delta I}\ \overset{?}{>}\ 0 \qquad (A.8)$$

In order for Inequality A.8 to be satisfied for positive values of K, it must at least be true that

$$[\underline{E}-\underline{e}]^T\ \underline{\Delta I}\ \overset{?}{>}\ 0 \qquad (A.9)$$

Notice, however, that ΔI_1, the current flowing <u>into</u> the ith node, can be related to the node potentials by

$$-\Delta I_i = (g_{i1}\ g_{i2} \cdots g_{iN}) \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_N \end{bmatrix} - I_{Si} \qquad (A.10)$$

where $g_{i1} \cdots g_{iN}$ is the ith row of the conductance matrix $[G]$ for the network and I_{Si} is the equivalent current source feeding the ith node.

If we multiply Equation A.10 by -1 and write the result in matrix notation, we have

$$\underline{\Delta I} = -\ [G]\ \underline{e} + \underline{I_S} \qquad (A.11)$$

where

$$\underline{I_S} \triangleq \begin{bmatrix} I_{S1} \\ I_{S2} \\ \cdot \\ \cdot \\ \cdot \\ I_{SN} \end{bmatrix} \qquad (A.12)$$

Now observe that the equilibrium solution vector $\underline{E}$ by definition satisfies

$$\underline{I_S} = [G]\ \underline{E} \qquad (A.13)$$

If we substitute Equation A.13 into A.11 we find

$$\underline{\Delta I} = - [G]\, \underline{e} + [G]\underline{E} = [G][\underline{E}-\underline{e}] \qquad (A.14)$$

If we then substitute Equation A.14 into Inequality A.9, we obtain

$$[\underline{E}-\underline{e}]^T\, \underline{\Delta I} = [\underline{E}-\underline{e}]^T[G][\underline{E}-\underline{e}] > 0 \qquad (A.15)$$

The term on the left of Inequality A.15 is a quadratic form of the conductance matrix $[G]$. But since the conductance matrix for any linear amnesic network is positive definite,[14] Inequality A.15 will be satisfied for all vectors $[\underline{E}-\underline{e}] \neq \underline{0}$.[15] Now examining Inequality A.8, we see that it will be satisfied if we require

$$K < \frac{2[\underline{E}-\underline{e}]^T \underline{\Delta I}}{\underline{\Delta I}^T\, \underline{\Delta I}} \quad \triangleq V_m \qquad (A.16)$$

Let us now return to Inequality A.7. We will assume K to be negative and show that this leads to a contradiction. If K is negative, dividing Inequality A.7 by K yields

$$2[\underline{E}-\underline{e}]^T\, \underline{\Delta I} - K\, \underline{\Delta I}^T\, \underline{\Delta I} < 0 \qquad (A.17)$$

But negative values of K clearly cannot satisfy Inequality A.17. Hence K must be positive and upper bounded by Inequality A.16.

APPENDIX B

RELATION OF CURRENT ERROR TOLERANCE TO TOLERANCE ON NODE POTENTIALS

1. STATEMENT OF PROBLEM:

It is required that

$$\sum_{i=1}^{N} (E_i - e_i)^2 = (\underline{E}-\underline{e})^T (\underline{E}-\underline{e}) < \epsilon \qquad (B.1)$$

where $\underline{E}-\underline{e}$ is an N-dimensional vector with components $E_i - e_i$ $i=1, 2, \ldots, N$, and where ϵ is a small positive number. Let $\underline{\Delta I}$ be related to $\underline{E}-\underline{e}$ by

$$\underline{\Delta I} = [G](\underline{E}-\underline{e}) \qquad (B.2)$$

then it is desired to find a value $\delta = \delta(\epsilon)$ such that

$$\underline{\Delta I}^T \underline{\Delta I} = \sum_{i=1}^{N} (\Delta I_i)^2 < \delta \qquad (B.3)$$

implies Inequality B.1.

2. SOLUTION

Let the norm of any vector $\underline{x}$ be defined by

$$\|\underline{x}\| \triangleq \underline{x}^T \underline{x} \qquad (B.4)$$

The norm of a matrix $[A]$ is then defined by[11]

$$\|A\| \triangleq \max_{\|x\| = 1} \| [A] \underline{x} \| \quad . \qquad (B.5)$$

It is a property of the norm that if[11]

$$\underline{y} = [A] \underline{x} \qquad (B.6)$$

then

$$\|\underline{y}\| \leq \|A\| \cdot \|\underline{x}\| \qquad (B.7)$$

Now if both sides of Equation B.2 are pre-multiplied by $[G^{-1}]$ and Inequality B.7 is applied then the result is

$$\|\underline{E} - \underline{e}\| < \|G^{-1}\| \cdot \|\underline{\Delta I}\| \tag{B.8}$$

From Inequality B.3 and Equation B.4 the norm of $\underline{\Delta I}$ is upper bounded by δ. If δ is taken to be

$$\delta = \frac{\epsilon}{\|G^{-1}\|} \tag{B.9}$$

then Equations B.4 and B.9 and Inequalities B.3 and B.8 yield the desired result Inequality B.1.

APPENDIX C

PROOF OF PARASITIC EIGENFUNCTIONS

The network to be analyzed is shown in Figure C.1.

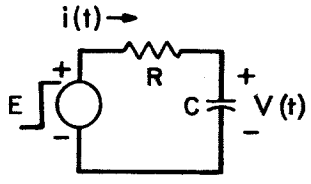


Fig. C.1 Series RC Circuit

The differential equation for the network is

$$i(t) = C\frac{dv(t)}{dt} = \frac{E-v(t)}{R} \qquad t \geq 0 \tag{C.1}$$

The solution of Equation C.1 is

$$v(t) = E(1-e^{-\frac{t}{RC}}) \quad t \geq 0 \tag{C.2}$$

Simpson's Rule when applied to the capacitor I-V relation yields

$$v_{n+1} = v_{n-1} + \frac{\Delta t}{3c}\left[i_{n-1} + 4i_n + i_{n+1}\right] \tag{C.3}$$

where the subscript indicates the point in time at which the voltage or current is to be evaluated. Equation C.1 yields

$$i_k = \frac{E-v_k}{R} \tag{C.4}$$

Putting this in Equation C.3 yields

$$v_{n+1} = v_{n-1} + \frac{\Delta t}{3CR}\left[6E - v_{n-1} - 4v_n - v_{n+1}\right] \tag{C.5}$$

or

$$v_{n+1} = v_{n-1} + \frac{Ah}{3}\left[v_{n-1} + 4v_n + v_{n+1}\right] - 2AEh \tag{C.6}$$

where

$$A \triangleq -\frac{1}{RC} \tag{C.7}$$

$$h \triangleq \Delta t \tag{C.8}$$

Regrouping terms in Equation C.6 yields

$$(1 - \frac{Ah}{3})\, v_{n+1} - \frac{4Ah}{3}\, v_n - (1 + \frac{Ah}{3})\, v_{n-1} + 2AEh = 0 \qquad (C.9)$$

Consider now the homogenious equation (E = 0)

$$(1 - \frac{Ah}{3})\, v_{n+1} - \frac{4Ah}{3}\, v_n - (1 + \frac{Ah}{3})\, v_{n-1} = 0 \qquad (C.10)$$

Let $V_k = \beta^k$ then Equation C.10 becomes

$$(1 - \frac{Ah}{3})\, \beta^{n+1} - \frac{4Ah}{3}\, \beta^n - (1 + \frac{Ah}{3})\, \beta^{n-1} = 0 \qquad (C.11)$$

Multiplying through by β^{-n+1} yields the characteristic equation

$$(1 - \frac{Ah}{3})\, \beta^2 - \frac{4Ah}{3}\, \beta - (1 + \frac{Ah}{3}) = 0 \qquad (C.12)$$

The solution of Equation C.9 will be of the form

$$v_n = c_o \beta_o^n + c_1 \beta_1^n + E \qquad (C.13)$$

where β_o and β_1 are the roots of the characteristic Equation C.12. Solving Equation C.12 for the roots yields

$$\beta = \left[\frac{2Ah}{3} \pm \sqrt{(\frac{2Ah}{3})^2 + 1 - (\frac{Ah}{3})^2}\right] \frac{1}{1 - \frac{Ah}{3}}$$

$$= \left[\frac{2Ah}{3} \pm \sqrt{1 + 3\,(\frac{Ah}{3})^2}\right] \frac{1}{1 - \frac{Ah}{3}} \qquad (C.14)$$

For small values of h, the square root can be represented by the first two terms in the binomial expansion

$$\beta \approx \left[\frac{2Ah}{3} \pm 1 \pm \frac{3}{2}\,(\frac{Ah}{3})^2\right] \frac{1}{1 - \frac{Ah}{3}} \qquad (C.15)$$

Expanding the fraction in Equation C.15 in a power series, multiplying

and neglecting terms of second order or higher yields

$$\beta \approx \pm 1 + \frac{2Ah}{3} \pm \frac{Ah}{3} \qquad (C.16)$$

or

$$\beta_o \approx 1 + Ah \qquad (C.17.a)$$

$$\beta_1 \approx -1 + \frac{1}{3} Ah \qquad (C.17.b)$$

From Equation C.13 the solution will be

$$v_n = C_o(1 + Ah)^n + (-1)^n C_1(1 - \frac{1}{3} Ah)^n + E \qquad (C.18)$$

The two terms to the power n behave approximately exponentially; hence, the solution is

$$\begin{aligned} v_n &\approx C_o e^{Ahn} + (-1)^n C_1 e^{-\frac{1}{3}Ahn} + E \\ &= C_o e^{\frac{t_n}{RC}} + (-1)^n C_1 e^{+\frac{tn}{3RC}} + E \end{aligned} \qquad (C.19)$$

If there is any initial error in the solution, C_1 will not be zero and hence a parasitic solution alternating in sign and exponentially increasing with a time constant of three times that for the RC circuit will be present.

APPENDIX D

PROGRAMS

1. A NOTE ON THE AED-0 LANGUAGE

The circuit analysis programs were written entirely in AED-0, the Project MAC version of ALGOL-60. The general structure of the AED language is identical to that of ALGOL as described by McCraken[18] except for transliterations (e.g. the $, replaces the semicolon as the end-of-statement mark). However, data structure features are available which are not included in ALGOL. Those features of the data structure used in the circuit analysis programs will be briefly described here.

It is possible to define a variable called a "pointer" which contains the address of another variable in a list. If P is a pointer to a list and V is a variable in the list, then V is referenced by the statement V(P). It is possible for the list itself to contain pointers to other lists which in turn may contain pointers to still other lists. In our programs variables in the P lists are referenced by pointers (P.LIST) in the junction boxes which in turn may be referenced by some index pointer X. Hence, a variable, say P1, in the P list would be referenced by a statement of the form

P1(P.LIST(X)).

So called "packed components" which form only parts of a computer word are referenced by pointers in the same way. A complete description of the structure of the lists and pointers is given in Reference 24.

2. PROGRAM FLOW CHARTS AND LISTINGS

DEFINITION OF VARIABLES AND PROCEDURES USED IN THE PROGRAMS

Variables Common to all Programs

COORD1 - not used
COORD2 - not used
DI - sum of currents at node
DUMMY - needed for display program
E - node potential
FIRST - pointer to first junction box in string
J1 - array of pointers to junction boxes
J2 - array of pointers to junction boxes
LIST - array of pointers to the p lists
MOST - total number of nodes (excluding node 0)
N - total number of time points
NC - pointer to opposite node
NEXT - pointer to next junction box in string
NODE - array of pointers to the node beads
N1 - node number
N2 - node number
P.LIST - pointer to p list
P1 - value of resistor in amnesic model
P2 - amplitude of voltage source in branch
P3 - value of voltage source in amnesic model
P4 - value of current source in amnesic model
P5 - frequency of source in branch
P6 - value of element in branch
P7 - auxiliary storage
P8 - auxiliary storage
SIGN - indicates direction of source in branch
SOTYPE - source type
T - array of time points
TP - indicates end of string
TYPE - branch type (RV, CI, etc.)
V - array of node potentials
x - index pointer

CIRCAL

Variables

CONTENTS - obtains variable in a location specified by a pointer
I - index
NX - pointer to data read with RWORD
TABLE - array of commands
TY - argument of RWORD

Procedures

DECODE	-	converts BCD number into integer form
PEKCHR	-	examines on line BCD item without "reading"
RT	-	character table used by RWORD
RWORD	-	reads BCD items on line
SETCT	-	sets characters in character table
SETHOW	-	used by RWORD

TOPO + INPT

Variables

T1	-	branch type (RV, CI, etc.)
T2	-	source type (sin, cos, u)
V1	-	value of element in branch
V2	-	amplitude of source in branch
V3	-	frequency of source in branch

Procedures

FREZ	-	obtains blocks of storage registers from free storage

DYSOL

Variables

DT	-	increment between successive time points
ELEMENT	-	bits 0 through 5 in TYPE
J	-	index
L	-	index
Q	-	argument of MNIP
SAMPLE	-	value of source at a particular point in time
SOURCE	-	bits 6 through 11 in TYPE
VR	-	voltage across resistor in capacitor model; capacitor through resistor in inductor model

MNIP

Variables

C	-	dummy argument used in procedure U
DISAVED	-	used for saving sum of currents at node
ER	-	value of procedure ERR
ERR1	-	temporary variable used in procedure U
ERSV	-	saved value of the current error

ERX	-	value of the current error
ESAVED	-	used for saving node potentials
KC	-	intermediate value of convergence constant
KO	-	optimum value of convergence constant
K1	-	value of convergence constant above upper limit
K2	-	value of convergence constant below upper limit
L	-	index
NIT	-	number of iterations
Q	-	time point index (also used to indicate errors)
UC	-	ordinate at KC
UG	-	value of procedure U
U1	-	ordinate at K1
U2	-	ordinate at K2

Procedures

ERR	-	computes current error
U	-	computes points on the U(K) curve

.PRNT

Variables

F	-	array of voltages to be plotted
HEAD	-	bits 0 through 11 in last word of NAME
J	-	index
MA	-	initial point to be plotted
MB	-	total number of points to be plotted
NAME	-	BCQ pointer to array of graph identification
NUMBER	-	bits 6 through 17 of BCQ-converted integer
Q	-	command (PLOT, PRINT, DISPLY)
TAIL	-	bits 18 through 29 in last word of NAME
TS	-	array of time points to be plotted

Procedures

NPLOT	-	plots graph on typewriter
NUMTOQ	-	converts an integer to BCQ form
XYPLOT	-	plots graph on CRT display

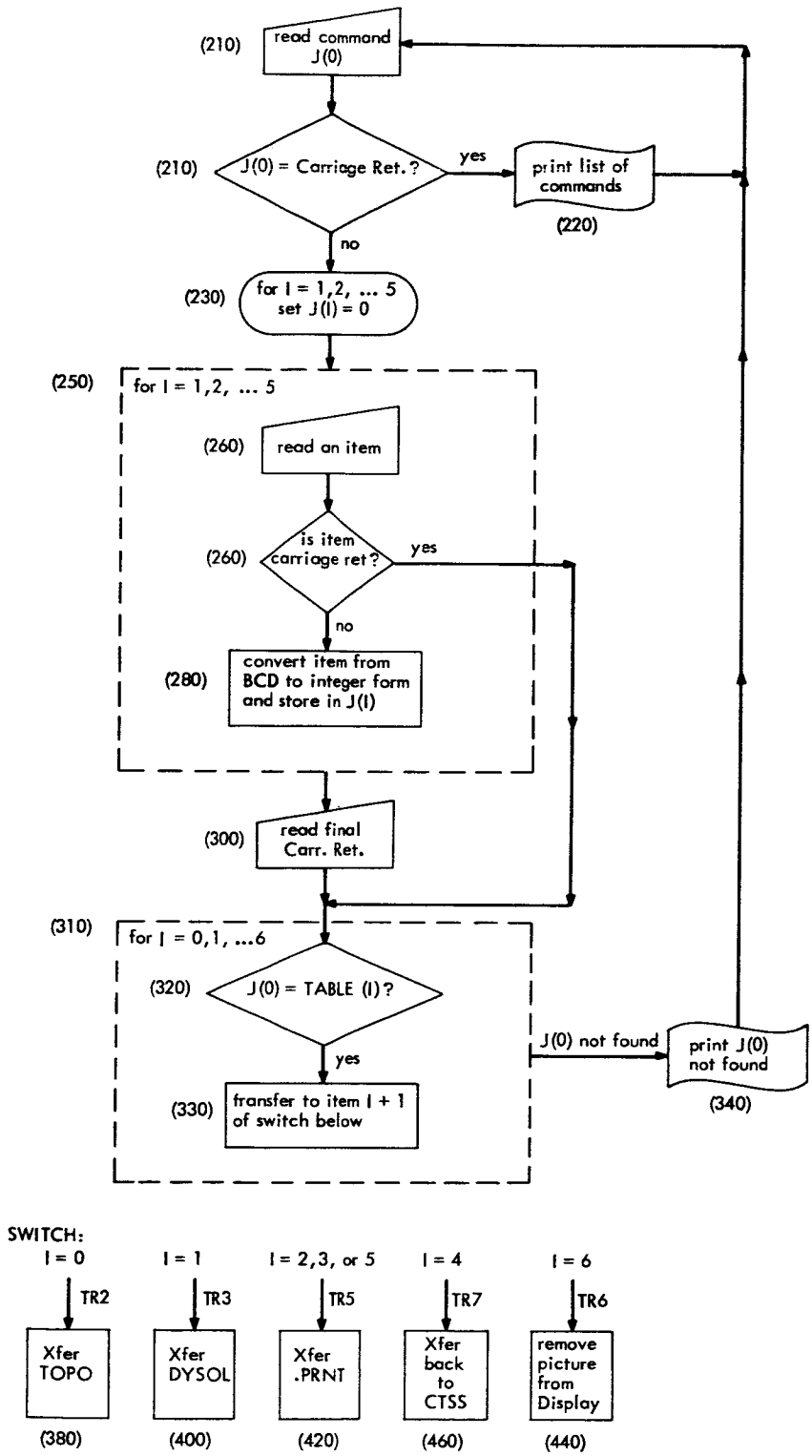
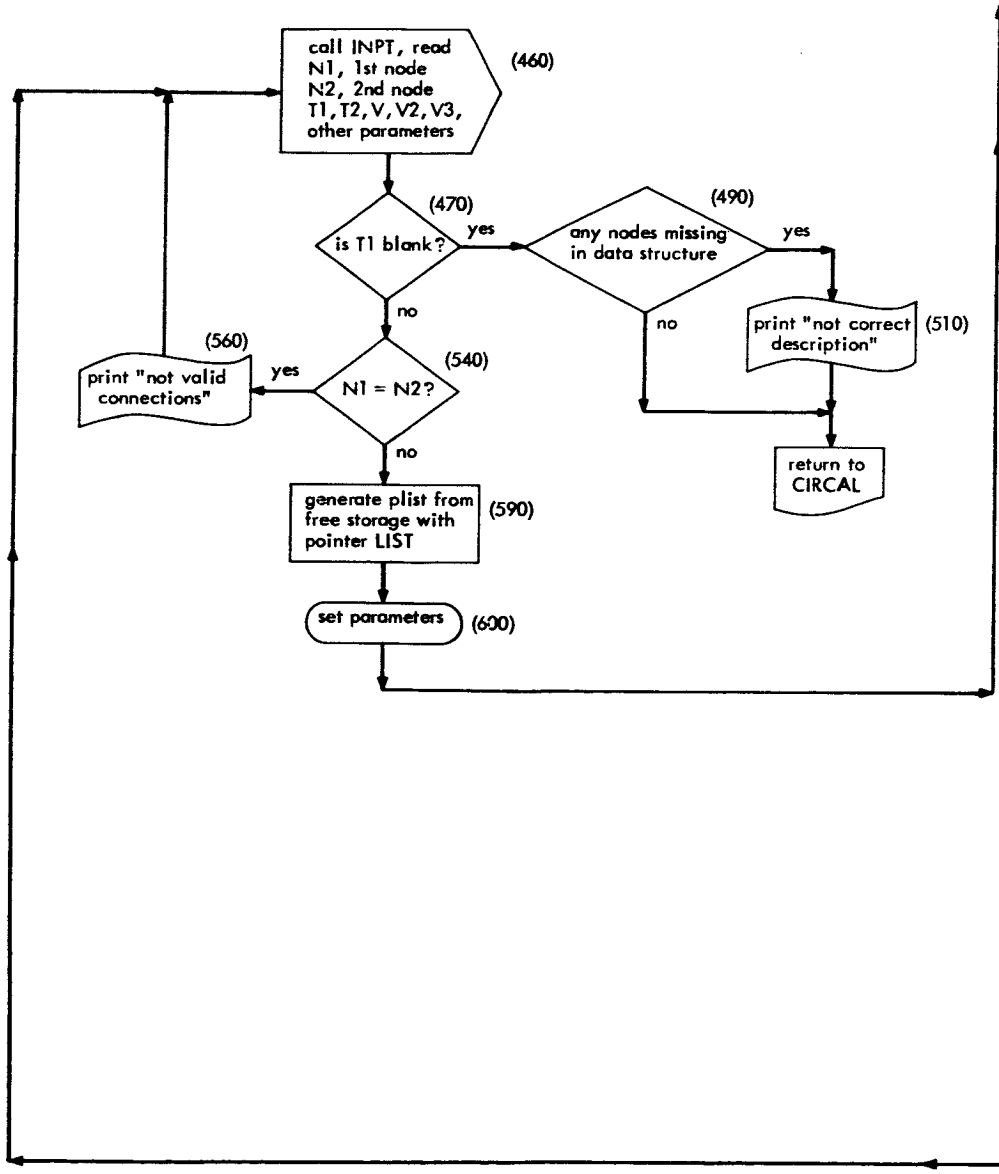


Fig. D.1 Flow Chart for CIRCAL

```
          BEGIN                                                                  00010
          INTEGER I,NX,TY $,                                                     00020
          INTEGER COMPONENT CONTENTS $,                                          00030
          CONTENTS $=$ 0 $,                                                      00040
          INTEGER ARRAY TABLE(10),J(5) $,                                        00050
          INTEGER PROCEDURE RWORD,SETHOW,RT,SETCT,PEKCHR,DECODE,SGNON,           00060
           CHNCOM $,                                                             00070
          SWITCH DOSW = TR2,TR3,TR5,TR5,TR7,TR5,TR6 $,                           00080
          PRESET TABLE(0) = 314547646360C,214521433171C,474346636060C,          00090
           475131456360C,506431636060C,243162474370C,255121622560C $,           00100
          CR .BCDN. -60000100000055- $,                                          00110
          CS .BCDN. -60000100000073- $,                                          00120
          CT .BCDN. -60000100000074- $,                                          00130
          CU .BCDN. -60000100000034- $,                                          00140
          SETCT(CONTENTS(RT),LOC CR(1),0,0,17,0,0,0) $,                          00150
          SETCT(CONTENTS(RT),LOC CS(1),0,0,0,0,0,0) $,                           00160
          SETCT(CONTENTS(RT),LOC CT(1),0,0,0,0,0,0) $,                           00170
          SETCT(CONTENTS(RT),LOC CU(1),0,0,0,0,0,0) $,                           00180
          SETHOW(0,0,0) $,                                                       00190
QRS $     PRINT FO4 $,                                                           00200
          IF (J(0) = CONTENTS(RWORD(RT,TY))) EQL 556060606060C                   00210
          THEN GOTO TR1 $,                                                       00220
          FOR I = 1 STEP 1 UNTIL 5                                               00230
          DO J(I) = 0 $,                                                         00240
          FOR I = 1 STEP 1 UNTIL 5                                               00250
          DO   IF CONTENTS(NX = RWORD(RT,TY)) EQL 556060606060C                  00260
               THEN GOTO QRZ                                                     00270
               ELSE J(I) = DECODE(NX) $,                                         00280
          IF CONTENTS(PEKCHR(RT)) EQL 556060606060C                              00290
          THEN NX = RWORD(RT,TY) $,                                              00300
.QRZ $    FOR I = 0 STEP 1 UNTIL 6                                               00310
          DO   IF J(0) EQL TABLE(I)                                              00320
               THEN GOTO DOSW(I+1) $,                                            00330
          PRINT FO3,J(0) $,                                                      00340
          GOTO QRS $,                                                            00350
TR1 $     PRINT FO2 $,                                                           00360
          GOTO QRS $,                                                            00370
TR2 $     TOPO1() $,                                                             00380
          GOTO QRS $,                                                            00390
TR3 $     DYSOL4() $,                                                            0040C
          GOTO QRS $,                                                            00410
TR5 $     .PRNT1(J(0),J(2),J(3),J(4),J(5)) $,                                    00420
          GOTO QRS $,                                                            00430
TR6 $     SGNON(1) $,                                                            00440
          GOTO QRS $,                                                            00450
TR7 $     PRINT FO4 $,                                                           00460
          CHNCOM(1) $,                                                           00470
FO2 $     FORMAT (/16HTHE COMMANDS ARE//                                         00480
    5HINPUT/6HANALIZ/20HPRINT F(N1,N2) MB MA /                                   00490
    19HPLOT F(N1,N2) MB MA/21HDISPLY F(N1,N2) MB MA /                            00500
    5HERASE /4HQUIT //)  $,                                                      00510
FO3 $     FORMAT(/1H',A6,31H' IS NOT IN THE COMMAND TABLE. ,                     00520
 19HHIT CARRIAGE RETURN /                                                        00530
 37HFOR A LIST OF THE ALLOWABLE COMMANDS.  /)  $,                                00540
FO4 $     FORMAT(1H ) $,                                                         00550
          END FINI                                                               00560
```


call INPT, read
N1, 1st node
N2, 2nd node
T1, T2, V, V2, V3,
other parameters
(460)
is T1 blank?
(470)
yes
any nodes missing
in data structure
(490)
yes
print "not correct
description"
(510)
no
return to
CIRCAL
no
N1 = N2?
(540)
yes
print "not valid
connections"
(560)
no
generate plist from
free storage with
pointer LIST
(590)
set parameters
(600)

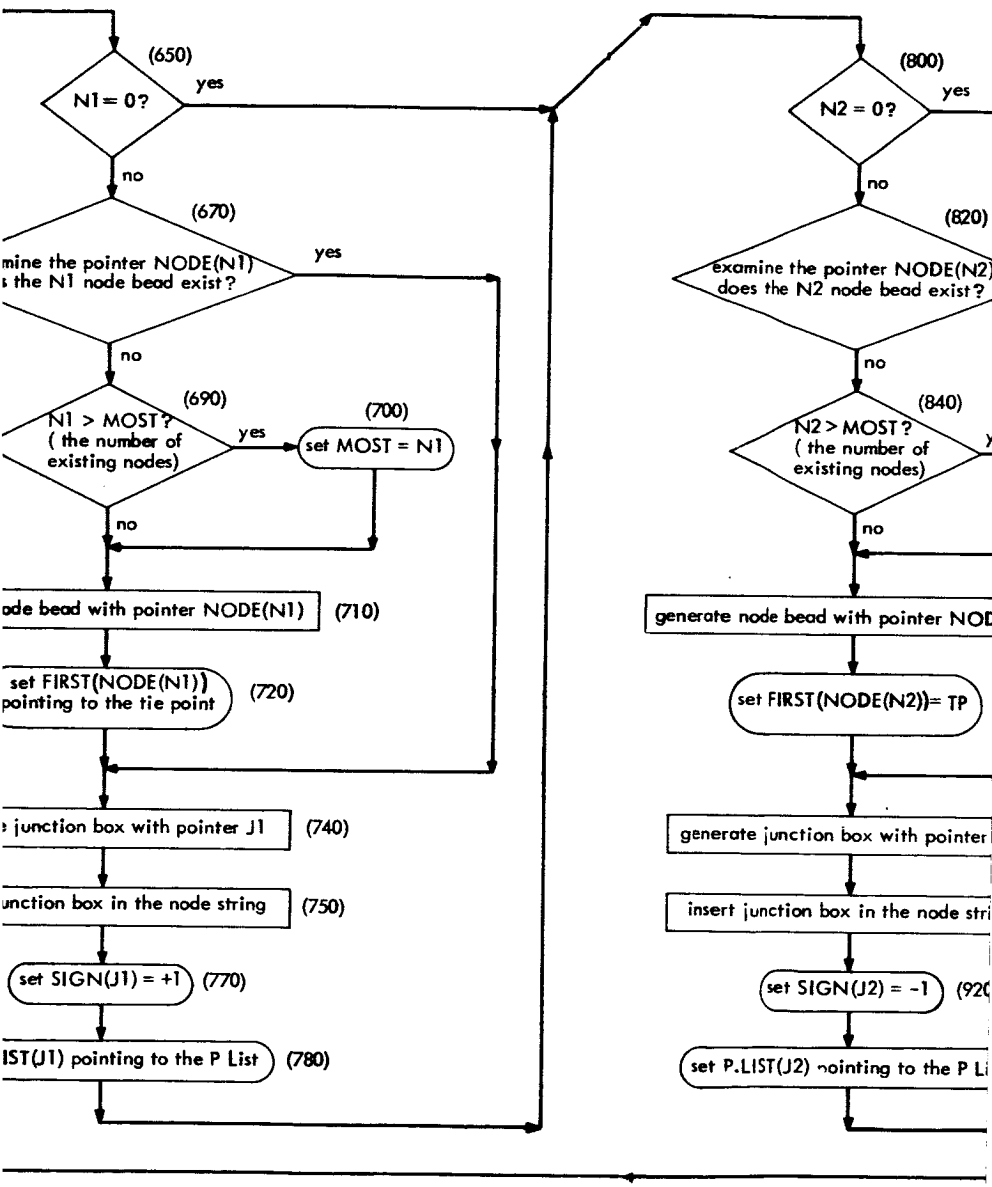
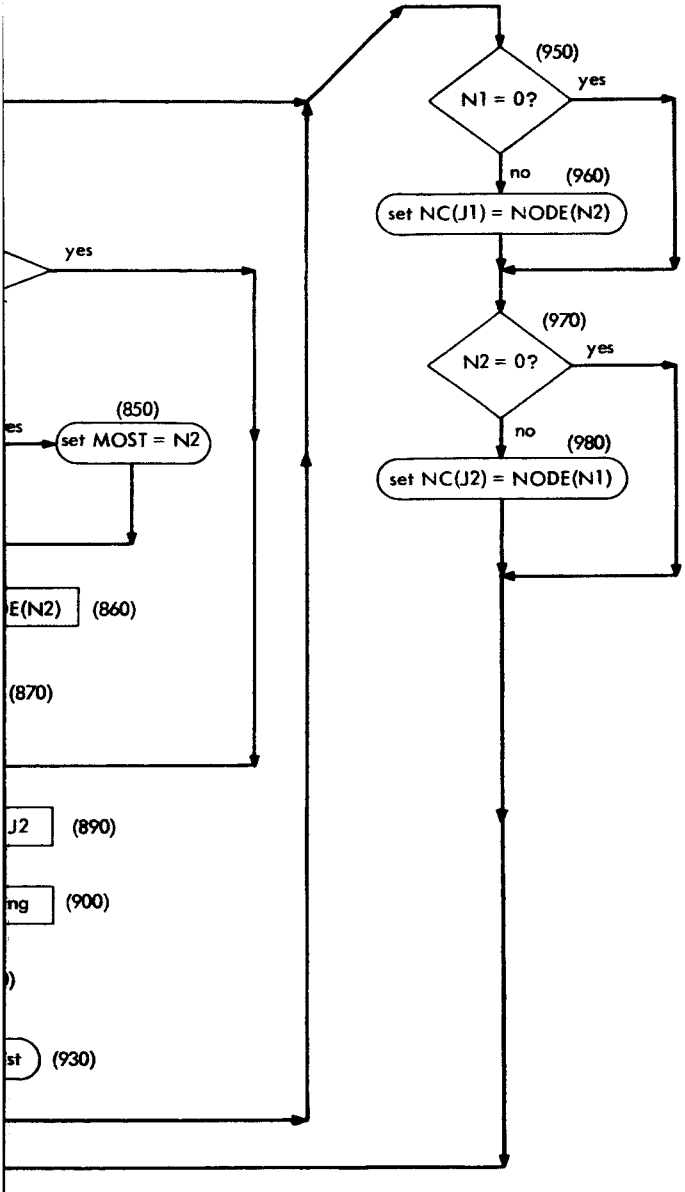


Fig. D.2 Flow Chart for TOPO

#2


(950)
N1 = 0?
yes
no
(960)
set NC(J1) = NODE(N2)
yes
(970)
N2 = 0?
yes
no
(980)
set NC(J2) = NODE(N1)
(850)
set MOST = N2
E(N2)
(860)
(870)
J2
(890)
ng
(900)
st
(930)

#3

```
BEGIN                                                                  00010
                                                                       00020
DEFINE PROCEDURE TOPO1() TOBE                                          00030
     BEGIN                                                             00040
     INTEGER PROCEDURE FREZ $,                                         00050
     REAL V1,V2,V3,EPS $,                                              00060
     REAL ARRAY T(200),V(4200) $,                                      00070
     INTEGER ARRAY DUMMY(150) $,                                       00080
     INTEGER N1,N2,T1,T2,L,MOST,TP,N,J $,                              00090
     INTEGER ARRAY NODE(21) $,                                         00100
     REAL COMPONENT E,DI $,                                            00110
     INTEGER COMPONENT FIRST,COORD1,COORD2 $,                          00120
     E $=$ 0 $,                                                        00130
     DI $=$ 1 $,                                                       00140
     FIRST $=$ 2 $,                                                    00150
     COORD1 $=$ 3 $,                                                   00160
     COORD2 $=$ 4 $,                                                   00170
     INTEGER ARRAY LIST(50) $,                                         00180
     INTEGER COMPONENT TYPE,SOTYPE $,                                  00190
     REAL COMPONENT P1,P2,P3,P4,P5,P6,P7,P8 $,                         00200
     TYPE $=$ 0 $,                                                     00210
     SOTYPE $=$ 1 $,                                                   00220
     P1 $=$ 2 $,                                                       00230
     P2 $=$ 3 $,                                                       00240
     P3 $=$ 4 $,                                                       00250
     P4 $=$ 5 $,                                                       00260
     P5 $=$ 6 $,                                                       00270
     P6 $=$ 7 $,                                                       00280
     P7 $=$ 8 $,                                                       00290
     P8 $=$ 9 $,                                                       00300
     INTEGER ARRAY J1(50),J2(50) $,                                    00310
     INTEGER COMPONENT NEXT,NC,P.LIST $,                               00320
     REAL COMPONENT SIGN $,                                            00330
     NEXT $=$ 0 $,                                                     00340
     NC $=$ 1 $,                                                       00350
     P.LIST $=$ 2 $,                                                   00360
     SIGN $=$ 3 $,                                                     00370
     COMMON DUMMY,NODE,LIST,J1,J2,MOST,TP,EPS,V,T,N $,                 00380
     NODE(0) = FREZ(5) $,                                              00390
     E(NODE(0)) = 0. $,                                                00400
     MOST = 0 $,                                                       00410
     TP = 212121212121C $,                                             00420
     PRINT GO $,                                                       00430
     FOR L = 0 STEP 1 UNTIL 49                                         00440
     DO   BEGIN                                                        00450
SPEC $    INPT(N1,N2,T1,T2,V1,V2,V3) $,                               00460
          IF T1 EQL 606060606060C                                      00470
          THEN BEGIN                                                   00480
               FOR J = 0 STEP 1 UNTIL MOST                             00490
               DO   IF NODE(J) EQL 0                                   00500
                    THEN PRINT ALARM $,                                00510
               GOTO RETURN $,                                          00520
               END $,                                                  00530
          IF N1 EQL N2                                                 00540
          THEN BEGIN                                                   00550
               PRINT ERROR $,                                          00560
               GOTO SPEC $,                                            00570
```

```
                    END $,                                          00580
               LIST(L) = FREZ(10) $,                                00590
               P6(LIST(L)) = V1 $,                                  00600
               P2(LIST(L)) = V2 $,                                  00610
               P5(LIST(L)) = V3 $,                                  00620
               TYPE(LIST(L)) = T1 $,                                00630
               SOTYPE(LIST(L)) = T2 $,                              00640
               IF N1 NEQ 0                                          00650
               THEN BEGIN                                           00660
                    IF NODE(N1) EQL 0                               00670
                    THEN BEGIN                                      00680
                         IF N1 GRT MOST                             00690
                         THEN MOST = N1 $,                          00700
                         NODE(N1) = FREZ(5) $,                      00710
                         FIRST(NODE(N1)) = TP $,                    00720
                         END $,                                     00730
                    J1(L) = FREZ(4) $,                              00740
                    NEXT(J1(L)) = FIRST(NODE(N1)) $,                00750
                    FIRST(NODE(N1)) = J1(L) $,                      00760
                    SIGN(J1(L)) = +1. $,                            00770
                    P.LIST(J1(L)) = LIST(L) $,                      00780
                    END $,                                          00790
               IF N2 NEQ 0                                          00800
               THEN BEGIN                                           00810
                    IF NODE(N2) EQL 0                               00820
                    THEN BEGIN                                      00830
                         IF N2 GRT MOST                             00840
                         THEN MOST = N2 $,                          00850
                         NODE(N2) = FREZ(5) $,                      00860
                         FIRST(NODE(N2)) = TP $,                    00870
                         END $,                                     00880
                    J2(L) = FREZ(4) $,                              00890
                    NEXT(J2(L)) = FIRST(NODE(N2)) $,                00900
                    FIRST(NODE(N2)) = J2(L) $,                      00910
                    SIGN(J2(L)) = -1. $,                            00920
                    P.LIST(J2(L)) = LIST(L) $,                      00930
                    END $,                                          00940
               IF N1 NEQ 0                                          00950
               THEN NC(J1(L)) = NODE(N2) $,                         00960
               IF N2 NEQ 0                                          00970
               THEN NC(J2(L)) = NODE(N1) $,                         00980
               END $,                                               00990
ERROR $     FORMAT ( /24HTHAT'S NOT A CONNECTION.  /)  $,           01000
GO $        FORMAT (/24HTYPE BRANCH DESCRIPTION.  /)  $,            01010
ALARM $     FORMAT (/32HNETWORK NOT CORRECTLY DESCRIBED.            01020
     /) $, END $,                                                   01030
                                                                    01040
      END FINI                                                      01050
```

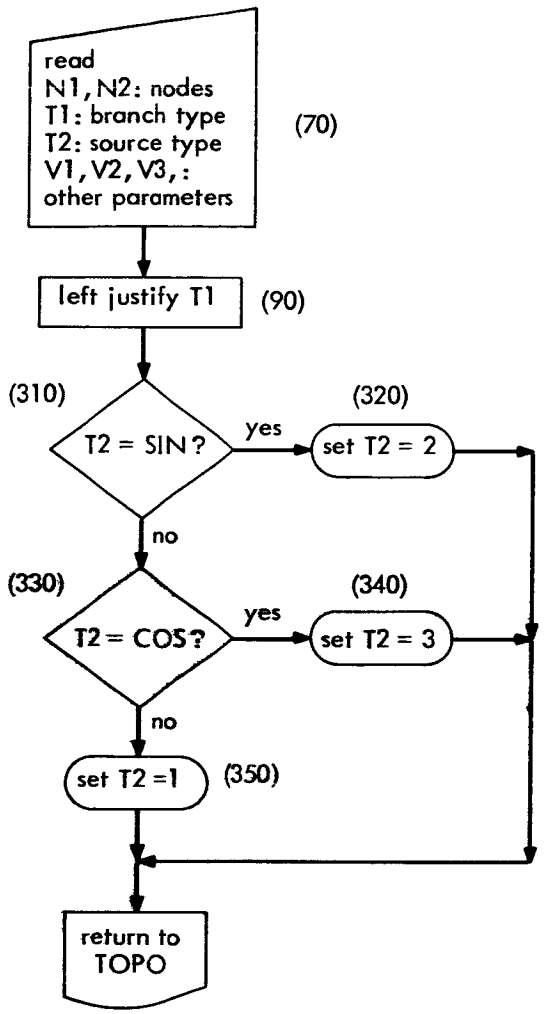
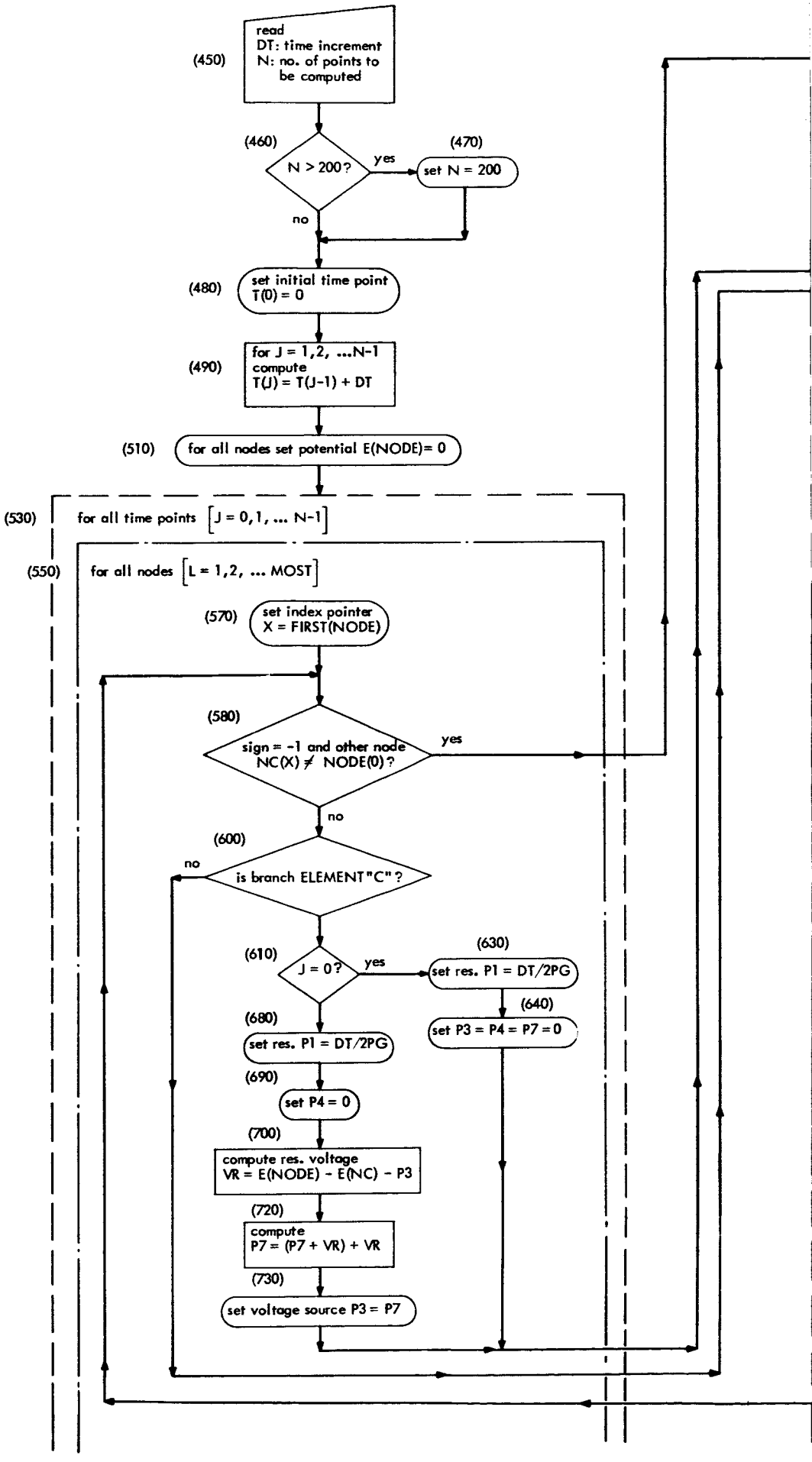


Fig. D.3 Flow Chart for INPT

```
          BEGIN                                                                  00010
                                                                                 00020
          DEFINE PROCEDURE INPT(N1,N2,T1,T2,V1,V2,V3) WHERE INTEGER N1,          00030
           N2,T1,T2 $,                                                           00040
          REAL V1,V2,V3 $, TOBE                                                  00050
               BEGIN                                                             00060
               READ LINE,N1,N2,T1,V1,V2,T2,V3 $,                                 00070
LINE $         FORMAT (2I2,A6,2F8.0,A6,F8.0 )  $,                               00080
               IF T1 EQL 606060605131C                                           00090
               THEN T1 = 513160606060C                                           00100
               ELSE IF T1 EQL 606060605165C                                      00110
                    THEN T1 = 516560606060C                                      00120
                    ELSE IF T1 EQL 606060606051C                                 00130
                         THEN T1 = 516060606060C                                 00140
                         ELSE IF T1 EQL 606060602331C                            00150
                              THEN T1 = 233160606060C                            00160
                              ELSE IF T1 EQL 606060602365C                       00170
                                   THEN T1 = 236560606060C                       00180
                                   ELSE IF T1 EQL 606060606023C                  00190
                                        THEN T1 = 236060606060C                  00200
                                        ELSE IF T1 EQL 606060604331C             00210
                                             THEN T1 = 433160606060C             00220
                                             ELSE IF T1 EQL                      00230
                                              606060604365C                      00240
                                                  THEN T1 =                      00250
                                                   436560606060C                 00260
                                                  ELSE IF T1 EQL                 00270
                                                   606060606043C                 00280
                                                       THEN T1 =                 00290
                                                        436060606060C $,         00300
               IF T2 EQL 606060623145C                                           00310
               THEN T2 = 2                                                       00320
               ELSE IF T2 EQL 606060234662C                                      00330
                    THEN T2 = 3                                                  00340
                    ELSE T2 = 1 $,                                               00350
               END $,                                                            00360
                                                                                 00370
          END FINI                                                               00380
```


(450)
read
DT: time increment
N: no. of points to be computed
(460)
N > 200?
yes
(470)
set N = 200
no
(480)
set initial time point
T(0) = 0
(490)
for J = 1,2, ...N-1
compute
T(J) = T(J-1) + DT
(510)
for all nodes set potential E(NODE)= 0
(530)
for all time points [J = 0,1, ... N-1]
(550)
for all nodes [L = 1,2, ... MOST]
(570)
set index pointer
X = FIRST(NODE)
(580)
sign = -1 and other node
NC(X) ≠ NODE(0)?
yes
no
(600)
no
is branch ELEMENT"C"?
(610)
J = 0?
yes
(630)
set res. P1 = DT/2PG
(640)
set P3 = P4 = P7 = 0
(680)
set res. P1 = DT/2PG
(690)
set P4 = 0
(700)
compute res. voltage
VR = E(NODE) - E(NC) - P3
(720)
compute
P7 = (P7 + VR) + VR
(730)
set voltage source P3 = P7

#1

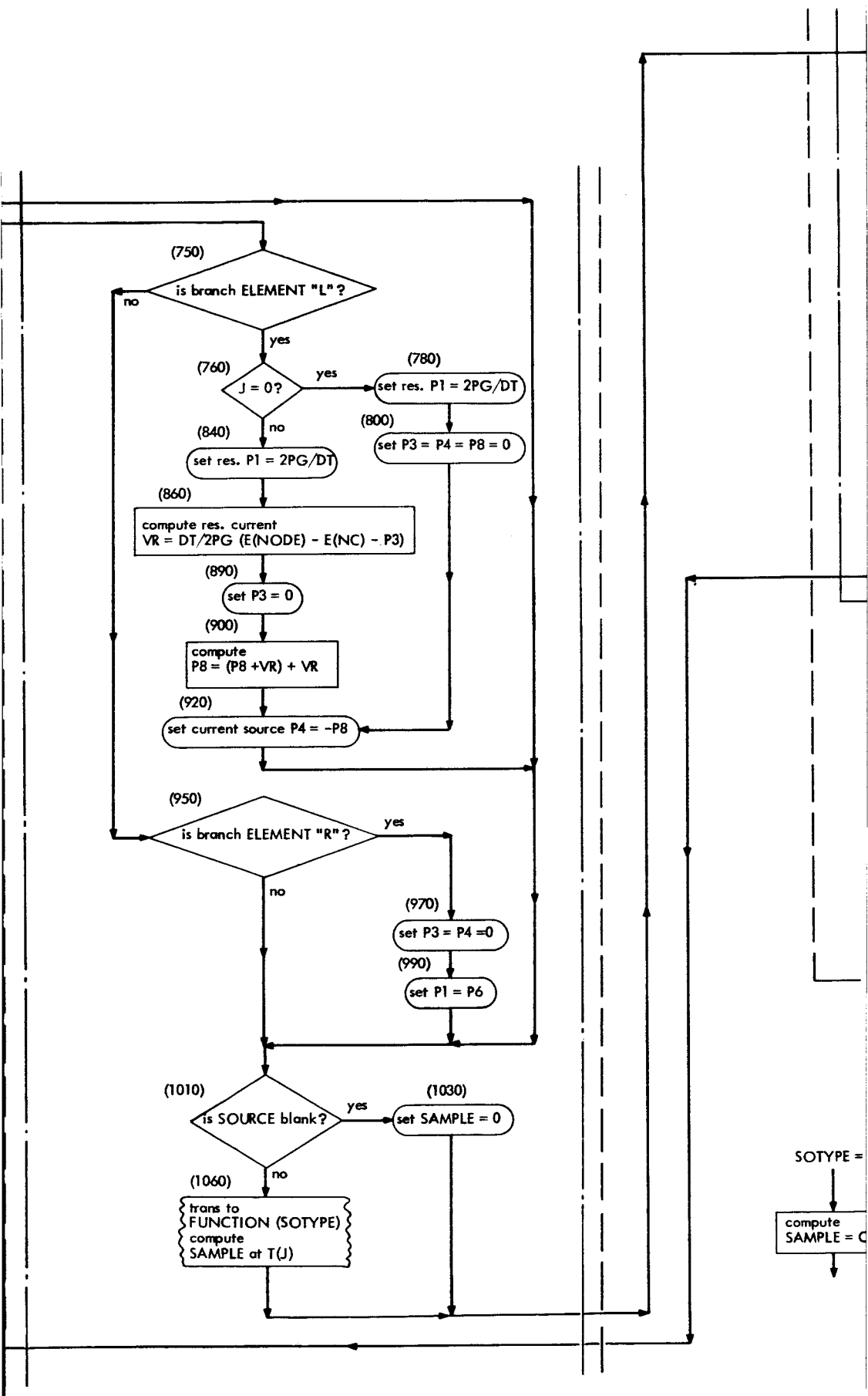
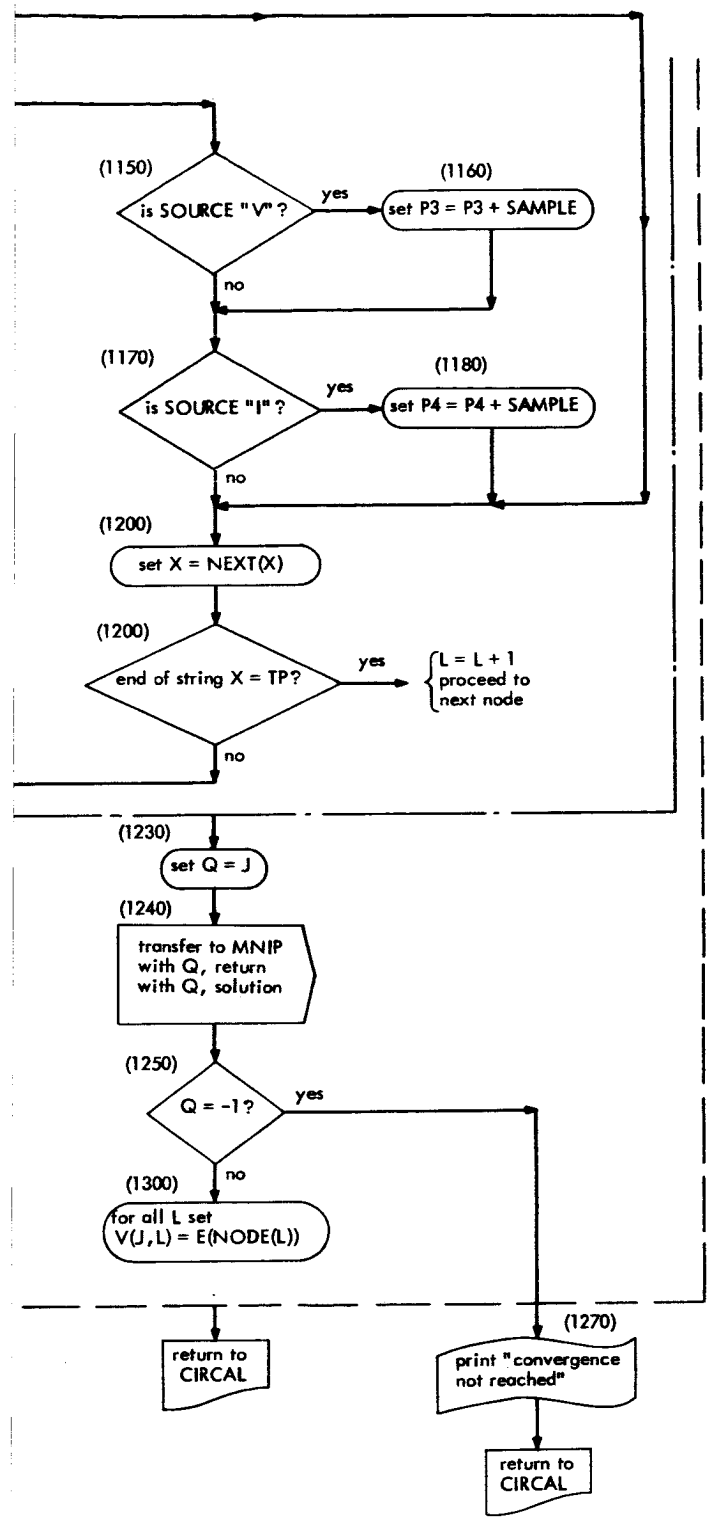


Fig. D.4 Flow Chart for DYSOL

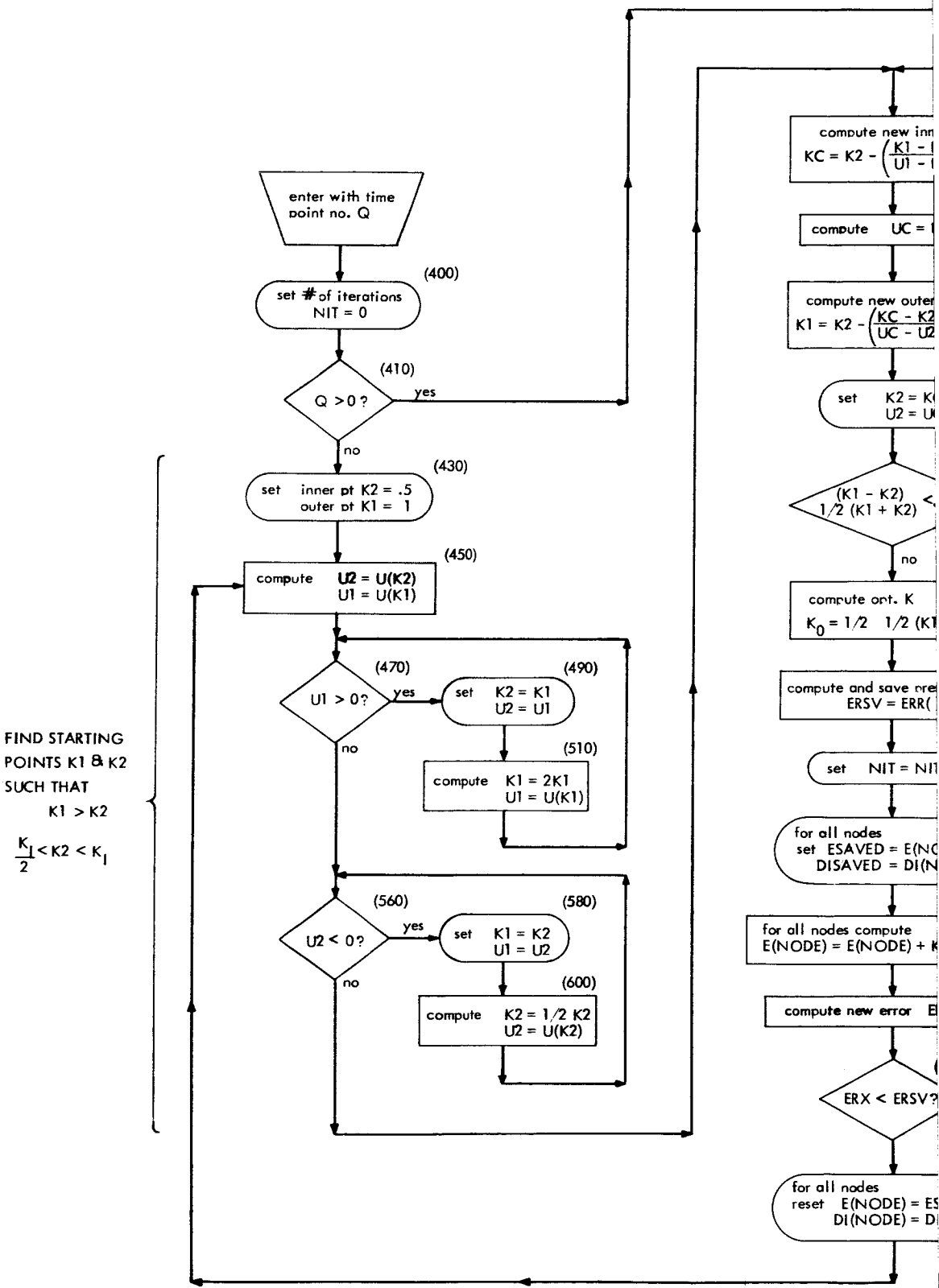
#2

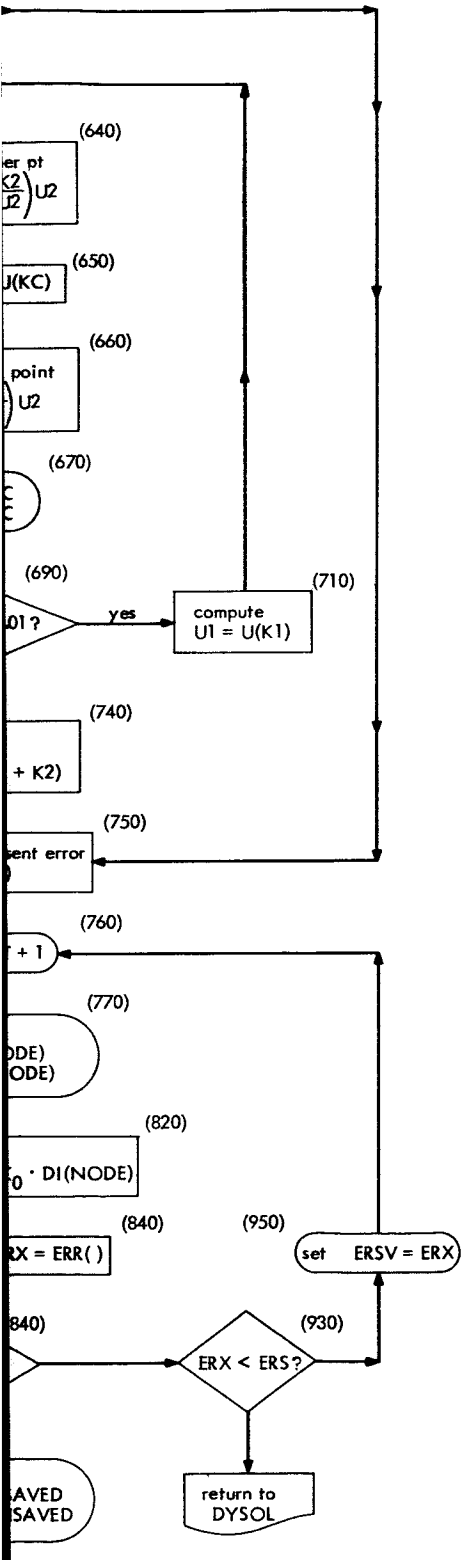

(1150)
is SOURCE "V"?
yes
(1160)
set P3 = P3 + SAMPLE
no
(1170)
is SOURCE "I"?
yes
(1180)
set P4 = P4 + SAMPLE
no
(1200)
set X = NEXT(X)
(1200)
end of string X = TP?
yes
L = L + 1
proceed to
next node
no
(1230)
set Q = J
(1240)
transfer to MNIP
with Q, return
with Q, solution
(1250)
Q = -1?
yes
no
(1300)
for all L set
V(J,L) = E(NODE(L))
return to
CIRCAL
(1270)
print "convergence
not reached"
return to
CIRCAL
SOTYPE = 2
SOTYPE = 3
(1070)
(1100)
(1130)
compute
SAMPLE = Sin
compute
SAMPLE=Const

```
BEGIN                                                                   00010
                                                                        00020
DEFINE PROCEDURE DYSOL4() TOBE                                          00030
     BEGIN                                                              00040
     REAL ARRAY T(200),V(4200) $,                                       00060
     REAL EPS,DT,SAMPLE,VR $,                                           00070
     INTEGER ARRAY DUMMY(150) $,                                        00080
     INTEGER MOST,TP,N,X,L,Q,J $,                                       00090
     INTEGER ARRAY NODE(21) $,                                          00100
     REAL COMPONENT E,DI $,                                             00110
     INTEGER COMPONENT FIRST,COORD1,COORD2 $,                           00120
     E $=$ 0 $,                                                         00130
     DI $=$ 1 $,                                                        00140
     FIRST $=$ 2 $,                                                     00150
     COORD1 $=$ 3 $,                                                    00160
     COORD2 $=$ 4 $,                                                    00170
     INTEGER ARRAY LIST(50) $,                                          00180
     INTEGER COMPONENT TYPE,SOTYPE,ELEMENT,SOURCE $,                    00190
     REAL COMPONENT P1,P2,P3,P4,P5,P6,P7,P8 $,                          00200
     PACK 77C30,30,SPECIAL COMPONENTS ELEMENT $,                        00210
     PACK 77C24,24,SPECIAL COMPONENTS SOURCE $,                         00220
     ELEMENT $=$ SOURCE $=$ 0 $,                                        00230
     TYPE $=$ 0 $,                                                      00240
     SOTYPE $=$ 1 $,                                                    00250
     P1 $=$ 2 $,                                                        00260
     P2 $=$ 3 $,                                                        00270
     P3 $=$ 4 $,                                                        00280
     P4 $=$ 5 $,                                                        00290
     P5 $=$ 6 $,                                                        00300
     P6 $=$ 7 $,                                                        00310
     P7 $=$ 8 $,                                                        00320
     P8 $=$ 9 $,                                                        00330
     INTEGER ARRAY J1(50),J2(50) $,                                     00340
     INTEGER COMPONENT NEXT,NC,P.LIST $,                                00350
     REAL COMPONENT SIGN $,                                             00360
     NEXT $=$ 0 $,                                                      00370
     NC $=$ 1 $,                                                        00380
     P.LIST $=$ 2 $,                                                    00390
     SIGN $=$ 3 $,                                                      00400
     COMMON DUMMY,NODE,LIST,J1,J2,MOST,TP,EPS,V,T,N $,                  00410
     SWITCH FUNCTION = U,S,C $,                                         00420
     Q = 0 $,                                                           00430
     PRINT LABLE $,                                                     00440
     READ INFO,DT,N $,                                                  00450
     IF N GRT 200                                                       00460
     THEN N = 200 $,                                                    00470
     T(0) = 0 $,                                                        00480
     FOR J = 1 STEP 1 UNTIL N-1                                         00490
     DO T(J) = T(J-1)+DT $,                                             00500
     FOR L = 0 STEP 1 UNTIL MOST                                        00510
     DO E(NODE(L)) = 0 $,                                               00520
     FOR J = 0 STEP 1 UNTIL N-1                                         00530
     DO   BEGIN                                                         00540
          FOR L = 1 STEP 1 UNTIL MOST                                   00550
          DO   BEGIN                                                    00560
               X = FIRST(NODE(L)) $,                                    00570
LP1 $          IF SIGN(X) NEQ +1 AND NC(X) NEQ NODE(0)                  00580
```

```
                    THEN GOTO CONTINUE $,                                   00590
                    IF ELEMENT(P.LIST(X)) EQL 23C                           00600
                    THEN IF J EQL 0                                         00610
                         THEN BEGIN                                         00620
                              P1(P.LIST(X)) = DT/P6(P.LIST(X))/2 $,         00630
                              P3(P.LIST(X)) = P4(P.LIST(X)) = P7(           00640
                               P.LIST(X)) = 0 $,                            00650
                              END                                           00660
                         ELSE BEGIN                                         00670
                              P1(P.LIST(X)) = DT/P6(P.LIST(X))/2 $,         00680
                              P4(P.LIST(X)) = 0 $,                          00690
                              VR = E(NODE(L))-E(NC(X))-P3(P.LIST(X)         00700
                               ) $,                                         00710
                              P7(P.LIST(X)) = P7(P.LIST(X))+2*VR $,         00720
                              P3(P.LIST(X)) = P7(P.LIST(X)) $,              00730
                              END                                           00740
                    ELSE IF ELEMENT(P.LIST(X)) EQL 43C                      00750
                         THEN IF J EQL 0                                    00760
                              THEN BEGIN                                    00770
                                   P1(P.LIST(X)) = 2*P6(P.LIST(X))/         00780
                                    DT $,                                   00790
                                   P3(P.LIST(X)) = P4(P.LIST(X)) =          00800
                                    P8(P.LIST(X)) = 0 $,                    00810
                                   END                                      00820
                              ELSE BEGIN                                    00830
                                   P1(P.LIST(X)) = 2*P6(P.LIST(X))/         00840
                                    DT $,                                   00850
                                   VR = (E(NODE(L))-E(NC(X))-P3(            00860
                                    P.LIST(X)))*DT/P6(P.LIST(X))/2          00870
                                    $,                                      00880
                                   P3(P.LIST(X)) = 0 $,                     00890
                                   P8(P.LIST(X)) = P8(P.LIST(X))+2*         00900
                                    VR $,                                   00910
                                   P4(P.LIST(X)) = -P8(P.LIST(X))           00920
                                    $,                                      00930
                                   END                                      00940
                         ELSE IF ELEMENT(P.LIST(X)) EQL 51C                 00950
                              THEN BEGIN                                    00960
                                   P3(P.LIST(X)) = P4(P.LIST(X)) =          00970
                                    0 $,                                    00980
                                   P1(P.LIST(X)) = P6(P.LIST(X)) $,         00990
                                   END $,                                   01000
                    IF SOURCE(P.LIST(X)) EQL 60C                            01010
                    THEN BEGIN                                              01020
                         SAMPLE = 0 $,                                      01030
                         GOTO CONTINUE $,                                   01040
                         END $,                                             01050
                    GOTO FUNCTION(SOTYPE(P.LIST(X))) $,                     01060
C $                 SAMPLE = P2(P.LIST(X))*COS(6.2831852*P5(P.LIST(         01070
                     X))*T(J)) $,                                           01080
                    GOTO SKIP $,                                            01090
S $                 SAMPLE = P2(P.LIST(X))*SIN(6.2831852*P5(P.LIST(         01100
                     X))*T(J)) $,                                           01110
                    GOTO SKIP $,                                            01120
U $                 SAMPLE = P2(P.LIST(X)) $,                               01130
SKIP $              SAMPLE = SAMPLE*SIGN(X) $,                              01140
                    IF SOURCE(P.LIST(X)) EQL 65C                            01150
                    THEN P3(P.LIST(X)) = P3(P.LIST(X))+SAMPLE               01160
```

```
                              ELSE IF SOURCE(P.LIST(X)) EQL 31C                  01170
                                   THEN P4(P.LIST(X)) = P4(P.LIST(X))+SAMPLE     01180
                                    $,                                           01190
CONTINUE $                    IF (X = NEXT(X)) NEQ TP                            01200
                              THEN GOTO LP1 $,                                   01210
                              END $,                                             01220
                         Q = J $,                                                01230
                         MNIP(Q) $,                                              01240
                         IF Q EQL -1                                             01250
                         THEN BEGIN                                              01260
                              PRINT BANANAS,J $,                                 01270
                              GOTO RETURN $,                                     01280
                              END $,                                             01290
THROUGH $                FOR L = 0 STEP 1 UNTIL MOST                            01300
                         DO V(J*21+L) = E(NODE(L)) $,                            01310
                         END $,                                                  01320
LABLE $             FORMAT (/35HTYPE TIME INCREMENT, NO. OF POINTS,    ,        01330
 19H ONE ITEM PER LINE.  /) $,                                                   01340
INFO $              FORMAT (F10.0/I3) $,                                        01350
BANANAS $           FORMAT (/25HCONVERGENCE NOT REACHED. ,I3,                    01360
      22H POINTS WERE COMPUTED. /) $,                                           01370
                    END $,                                                       01380
                                                                                 01390
          END FINI                                                               01400
```


FIND STARTING POINTS K1 & K2 SUCH THAT
K1 > K2
K1/2 < K2 < K1
enter with time point no. Q
(400)
set # of iterations NIT = 0
(410)
Q > 0 ?
yes
no
(430)
set inner pt K2 = .5
outer pt K1 = 1
(450)
compute U2 = U(K2)
U1 = U(K1)
(470)
U1 > 0?
yes
(490)
set K2 = K1
U2 = U1
no
(510)
compute K1 = 2K1
U1 = U(K1)
(560)
U2 < 0?
yes
(580)
set K1 = K2
U1 = U2
no
(600)
compute K2 = 1/2 K2
U2 = U(K2)
compute new inn
KC = K2 - (K1 - 1
U1 - 1
compute UC = 1
compute new outer
K1 = K2 - (KC - K2
UC - U2
set K2 = K(
U2 = U(
(K1 - K2)
1/2 (K1 + K2) <
no
compute opt. K
K0 = 1/2 1/2 (K1
compute and save pre
ERSV = ERR(
set NIT = NIT
for all nodes
set ESAVED = E(NO
DISAVED = DI(N
for all nodes compute
E(NODE) = E(NODE) + K
compute new error E
ERX < ERSV?
for all nodes
reset E(NODE) = ES
DI(NODE) = DI



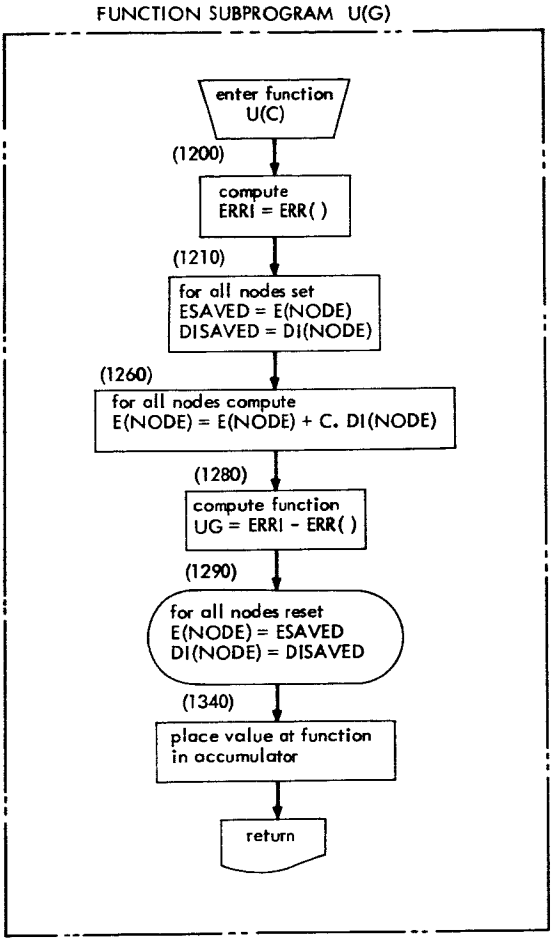
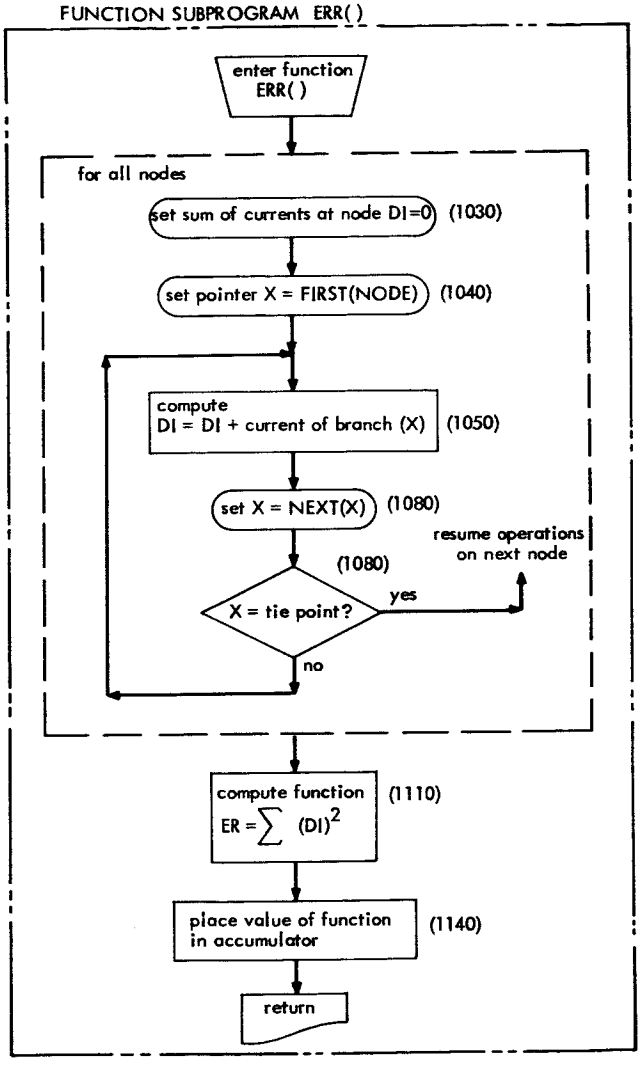


Fig. D.5 Flow Chart for MNIP


FUNCTION SUBPROGRAM ERR( )
enter function ERR( )
for all nodes
set sum of currents at node DI=0
(1030)
set pointer X = FIRST(NODE)
(1040)
compute DI = DI + current of branch (X)
(1050)
set X = NEXT(X)
(1080)
(1080)
X = tie point?
yes
resume operations on next node
no
compute function ER = ∑ (DI)^2
(1110)
place value of function in accumulator
(1140)
return

#3

```
          BEGIN                                                                  00010
                                                                                 00020
          DEFINE PROCEDURE MNIP(Q) WHERE INTEGER Q TOBE                          00030
               BEGIN                                                             00040
               REAL PROCEDURE ERR,U $,                                           00050
               REAL ER,EPS,K1,K2,KC,U1,U2,UC,KO,ERSV,ERX $,                      00060
               REAL ARRAY T(200),V(4200),ESAVED(21),DISAVED(21) $,               00070
               INTEGER MOST,TP,L,X,N,NIT $,                                      00080
               INTEGER ARRAY DUMMY(150) $,                                       00090
               INTEGER ARRAY NODE(21) $,                                         00100
               REAL COMPONENT E,DI $,                                            00110
               INTEGER COMPONENT FIRST,COORD1,COORD2 $,                          00120
               E $=$ 0 $,                                                        00130
               DI $=$ 1 $,                                                       00140
               FIRST $=$ 2 $,                                                    00150
               COORD1 $=$ 3 $,                                                   00160
               COORD2 $=$ 4 $,                                                   00170
               INTEGER ARRAY LIST(50) $,                                         00180
               INTEGER COMPONENT TYPE,SOTYPE $,                                  00190
               REAL COMPONENT P1,P2,P3,P4,P5,P6,P7,P8 $,                         00200
               TYPE $=$ 0 $,                                                     00210
               SOTYPE $=$ 1 $,                                                   00220
               P1 $=$ 2 $,                                                       00230
               P2 $=$ 3 $,                                                       00240
               P3 $=$ 4 $,                                                       00250
               P4 $=$ 5 $,                                                       00260
               P5 $=$ 6 $,                                                       00270
               P6 $=$ 7 $,                                                       00280
               P7 $=$ 8 $,                                                       00290
               P8 $=$ 9 $,                                                       00300
               INTEGER ARRAY J1(50),J2(50) $,                                    00310
               INTEGER COMPONENT NEXT,NC,P.LIST $,                               00320
               REAL COMPONENT SIGN $,                                            00330
               NEXT $=$ 0 $,                                                     00340
               NC $=$ 1 $,                                                       00350
               P.LIST $=$ 2 $,                                                   00360
               SIGN $=$ 3 $,                                                     00370
               COMMON DUMMY,NODE,LIST,J1,J2,MOST,TP,EPS,V,T,N $,                 00380
               PRESET EPS = .00001 $,                                            00390
               NIT = 0 $,                                                        00400
               IF Q GRT 0                                                        00410
               THEN GOTO SWING $,                                                00420
               K2 = .5 $,                                                        00430
               K1 = 1 $,                                                         00440
HEAD $         U2 = U(K2) $,                                                     00450
               U1 = U(K1) $,                                                     00460
REPEAT $       IF U1 GRT 0                                                       00470
               THEN BEGIN                                                        00480
                    K2 = K1 $,                                                   00490
                    U2 = U1 $,                                                   00500
                    K1 = 2*K1 $,                                                 00510
                    U1 = U(K1) $,                                                00520
                    GOTO REPEAT $,                                               00530
                    END                                                          00540
               ELSE                                                              00550
D.C. $              IF U2 LES 0                                                  00560
                    THEN BEGIN                                                   00570
```

```
                    K1 = K2 $,                                              00580
                    U1 = U2 $,                                              00590
                    K2 = K2/2 $,                                            00600
                    U2 = U(K2) $,                                           00610
                    GOTO D.C. $,                                            00620
                    END $,                                                  00630
CODA $      KC = K2-((K1-K2)/(U1-U2))*U2 $,                                 00640
            JC = U(KC) $,                                                   00650
            K1 = K2-((KC-K2)/(UC-U2))*U2 $,                                 00660
            K2 = KC $,                                                      00670
            U2 = UC $,                                                      00680
            IF 2*((K1-K2)/(K1+K2)) GRT .01                                  00690
            THEN BEGIN                                                      00700
                 U1 = U(K1) $,                                              00710
                 GOTO CODA $,                                               00720
                 END $,                                                     00730
            KO = (K1+K2)/4 $,                                               00740
SWING $     ERSV = ERR() $,                                                 00750
FINE $      NIT = NIT+1 $,                                                  00760
            FOR L = 1 STEP 1 UNTIL MOST                                     00770
            DO   BEGIN                                                      00780
                 ESAVED(L) = E(NODE(L)) $,                                  00790
                 DISAVED(L) = DI(NODE(L)) $,                                00800
                 END $,                                                     00810
            FOR L = 1 STEP 1 UNTIL MOST                                     00820
            DO E(NODE(L)) = E(NODE(L))+KO*DI(NODE(L)) $,                    00830
            IF ERSV LES (ERX = ERR())                                       00840
            THEN BEGIN                                                      00850
                 FOR L = 1 STEP 1 UNTIL MOST                                00860
                 DO   BEGIN                                                 00870
                      E(NODE(L)) = ESAVED(L) $,                             00880
                      DI(NODE(L)) = DISAVED(L) $,                           00890
                      END $,                                                00900
                 GOTO HEAD $,                                               00910
                 END $,                                                     00920
            IF ERX LES EPS                                                  00930
            THEN GOTO RETURN $,                                             00940
            ERSV = ERX $,                                                   00950
            GOTO FINE $,                                                    00960
                                                                            00970
            DEFINE REAL PROCEDURE ERR() TOBE                                00980
                 BEGIN                                                      00990
                 REAL ER $,                                                 01000
                 FOR L = 1 STEP 1 UNTIL MOST                                01010
                 DO   BEGIN                                                 01020
                      DI(NODE(L)) = 0 $,                                    01030
                      X = FIRST(NODE(L)) $,                                 01040
BRIDGE $              DI(NODE(L)) = DI(NODE(L))+(E(NC(X))+SIGN(X)*P3(       01050
                       P.LIST(X))-E(NODE(L)))/P1(P.LIST(X))+SIGN(X)*        01060
                       P4(P.LIST(X)) $,                                     01070
                      IF (X = NEXT(X)) NEQ TP                               01080
                      THEN GOTO BRIDGE $,                                   01090
                      END $,                                                01100
                 ER = 0 $,                                                  01110
                 FOR L = 1 STEP 1 UNTIL MOST                                01120
                 DO ER = ER+DI(NODE(L))*DI(NODE(L)) $,                      01130
                 ER = ER $,                                                 01140
                 END $,                                                     01150
```

```
      DEFINE REAL PROCEDURE U(C) WHERE REAL C TOBE                    01160
           BEGIN                                                      01170
           REAL ERR1,UG $,                                            01180
           ERR1 = ERR() $,                                            01190
           FOR L = 1 STEP 1 UNTIL MOST                                01200
           DO   BEGIN                                                 01210
                ESAVED(L) = E(NODE(L)) $,                             01220
                DISAVED(L) = DI(NODE(L)) $,                           01230
                END $,                                                01240
           FOR L = 1 STEP 1 UNTIL MOST                                01250
           DO E(NODE(L)) = E(NODE(L))+C*DI(NODE(L)) $,                01260
           UG = ERR1-ERR() $,                                         01270
           FOR L = 1 STEP 1 UNTIL MOST                                01280
           DO   BEGIN                                                 01290
                E(NODE(L)) = ESAVED(L) $,                             01300
                DI(NODE(L)) = DISAVED(L) $,                           01310
                END $,                                                01320
           UG = UG $,                                                 01330
           END $,                                                     01340
                                                                      01350
      END $,                                                          01360
                                                                      01370
END FINI                                                              01380
                                                                      01390
```

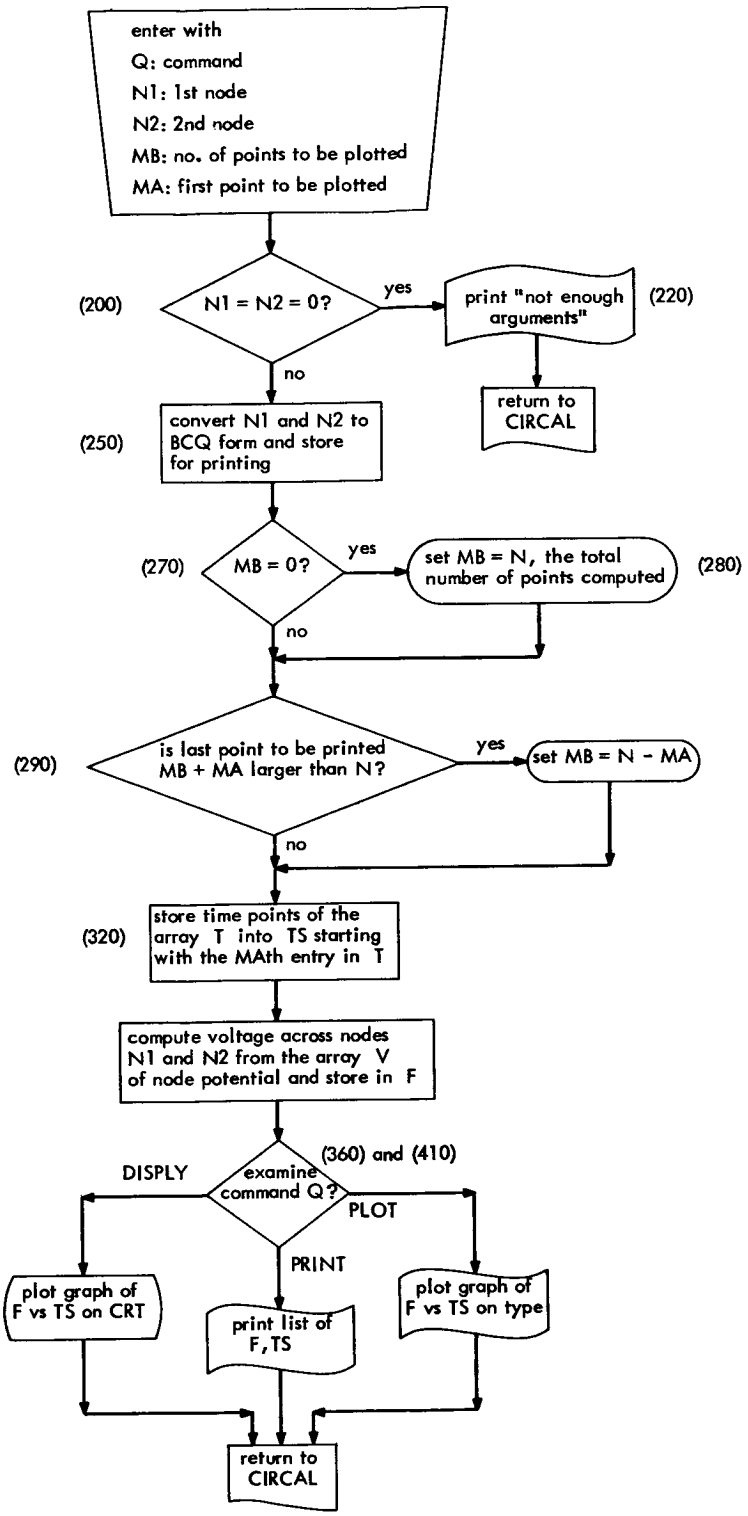


Fig. D.6 Flow Chart for PRNT

```
BEGIN                                                                   00010
                                                                        00020
DEFINE PROCEDURE .PRNT1(Q,N1,N2,MB,MA) WHERE INTEGER Q,N1,N2,           00030
 MB,MA TOBE                                                             00040
     BEGIN                                                              00050
     INTEGER PROCEDURE NUMTOQ $,                                        00060
     REAL ARRAY T(200),TS(200),F(200),V(4200) $,                        00070
     REAL EPS $,                                                        00080
     INTEGER NAME,N,MOST,TP,J $,                                        00090
     INTEGER ARRAY NODE(21),J1(50),J2(50),LIST(50),DUMMY(150)           00100
      $,                                                                00110
     INTEGER COMPONENT HEAD,TAIL,NUMBER $,                              00120
     HEAD $=$ TAIL $=$ 2 $,                                             00130
     NUMBER $=$ 0 $,                                                    00140
     PACK 7777C24,24,SPECIAL COMPONENTS HEAD $,                         00150
     PACK 7777C6,6,SPECIAL COMPONENTS TAIL $,                           00160
     PACK 7777C18,18,SPECIAL COMPONENTS NUMBER $,                       00170
     NAME .BCQ. /GRAPH OF V(   ,   )/ $,                                00180
     COMMON DUMMY,NODE,LIST,J1,J2,MOST,TP,EPS,V,T,N $,                  00190
     IF N1 EQL 0 AND N2 EQL 0                                           00200
     THEN BEGIN                                                         00210
          PRINT S0 $,                                                   00220
          GOTO RETURN $,                                                00230
          END $,                                                        00240
     HEAD(NAME) = NUMBER(NUMTOQ(N1)) $,                                 00250
     TAIL(NAME) = NUMBER(NUMTOQ(N2)) $,                                 00260
     IF MB EQL 0                                                        00270
     THEN MB = N $,                                                     00280
     IF MA+MB GRT N                                                     00290
     THEN MB = N-MA $,                                                  00300
     FOR J = 0 STEP 1 UNTIL MB-1                                        00310
     DO   BEGIN                                                         00320
          TS(J) = T(J+MA) $,                                            00330
          F(J) = V((J+MA)*21+N1)-V((J+MA)*21+N2) $,                     00340
          END $,                                                        00350
     IF Q EQL 474346636060C                                             00360
     THEN BEGIN                                                         00370
          NPLOT(F,TS,MB-1,NAME) $,                                      00380
          GOTO RETURN $,                                                00390
          END                                                           00400
     ELSE IF Q EQL 243162474370C                                        00410
          THEN BEGIN                                                    00420
               XYPLOT(F,TS,MB-1,NAME) $,                                00430
               GOTO RETURN $,                                           00440
               END $,                                                   00450
     PRINT S1,N1,N2 $,                                                  00460
     PRINT S2 $,                                                        00470
     FOR J = 0 STEP 2 UNTIL MB-1                                        00480
     DO PRINT S3,TS(J),F(J),TS(J+1),F(J+1) $,                           00490
     PRINT S4 $,                                                        00500
S0 $ FORMAT(/21HNOT ENOUGH ARGUMENTS. /)  $,                            00510
S1 $ FORMAT(/////35X,10HLIST OF V(,I2,1H,,I2,1H)  ///) $,               00520
S2 $ FORMAT(3X,2(8X,4HTIME,14X,8HFUNCTION,6X)  //)  $,                  00530
S3 $ FORMAT(4(1PE20.7)) $,                                              00540
S4 $ FORMAT(1H ///////)  $,                                             00550
     END $,                                                             00560
                                                                        00570
END FINI                                                                00580
```

REFERENCES

1. Allen, D.N. DeG., Relaxation Methods, McGraw-Hill Book Co., Inc., New York, 1954.

2. Birkhoff, G., and Diaz, S.B., "Nonlinear Network Problems", Quarterly of Applied Mathematics, Vol. 13, January, 1956, pp. 431-443.

3. Black, A.N., and Southwell, R.V., "Relaxation Methods Applied to Engineering Problems", Proceedings of the Royal Society of London A, Vol. 164, 1938.

4. Corbato, F.J., et al, Compatible Time-Sharing System, M.I.T. Press, Cambridge, Mass., 1963.

5. Dennis, J.B., Distributed Solution of Network Programming Problems, Internal Memorandum.

6. Dennis, J.B., Mathematical Programming and Electrical Networks, M.I.T. Press and John Wiley and Sons, Inc., New York, 1959.

7. Dertouzos, M.L., Threshold Logic: A Synthesis Approach, M.I.T. Press, Cambridge, Mass., 1965

8. Desoer, C.A., and Katzenelson, J., "Nonlinear RLC Networks", Bell System Technical Journal, Vol. 44, No. 1, January 1965, pp. 161-198.

9. Duffin, R.S., "Nonlinear Networks IIa", Bulletin of the American Mathematical Society, Vol. 53, October, 1947, pp. 963-971.

10. Duffin, R. S. "Nonlinear Networks IIb", Bulletin of the American Mathematical Society, Vol. 54, 1948, pp. 119-127.

11. Fadeeva, V.N., Computational Methods of Linear Algebra, Dover Publications, Inc., New York, 1959.

12. Fenves, Logcher, and Monch, STRESS Reference Manual, M.I.T. Press, Cambridge, Massachusetts.

13. Guillemin, E.A., Introductory Circuit Theory, John Wiley and Sons, Inc., New York, 1960.

14. Guillemin, E.A., Synthesis of Passive Networks, John Wiley and Sons, Inc., New York, 1962.

15. Hildebrand, F.B., Introduction to Numerical Analysis, McGraw-Hill Book Co., Inc., 1956.

REFERENCES (Cont.)

16. Hildebrand, F.B., Methods of Applied Mathematics, Prentice-Hall, Inc., Englewood Cliffs, 1961.

17. Katzenelson, J. and Seitelman, L.M., "An Iterative Method for Solution of Nonlinear Resistor Networks", To be Published.

18. Lee, H.B., Notes for M.I.T. Course 6.561.

19. Lee, H.B., Private Communication, August 3, 1965.

20. McCracken, D.D., A Guide to ALGOL Programming, John Wiley and Sons, Inc., New York, March 1964.

21. Meyer, C.S., "A Digital Computer Representation of the Linear Constant-Parameter Electric Network," M.S. Thesis, Department of Electrical Engineering, M.I.T., 1960.

22. Minty, G. "Solving Steady-State Nonlinear Networks of 'Monotone' Elements", IRE Transactions on Circuit Theory, June, 1961.

23. Roos, D. and Miller, C.L., The Internal Structure of COGO-90, Research Report R64-5, School of Engineering, M.I.T., February, 1964.

24. Ross, D.T., "AED-0 Programming Manual Preliminary Release 1 through 4," Internal Memorandum, 1964.

25. Scarsborough, J.B., Numerical Mathematical Analysis, John Hopkins Press, Baltimore, 1958.

26. Southwell, R.V., "Stress Calculation in Frameworks by The Method of Systematic Relaxation of Constraints", I and II., Proceedings of the Royal Society of London A, Vol. 164, 1938.

27. Zadek, L.A. and Desoer, C.A., Linear System Theory: The State Space Approach, McGraw-Hill, New York.

28. Zimmerman, H.J., and Mason, S.J., Electronic Circuit Theory, John Wiley and Sons, Inc., New York, 1960.